Fundamenta Informaticae 71 (2006) 15–26 15
IOS Press

Architectural Principles and Scheduling Strategies for Computing Agent Systems

Marek Grochowski, Robert Schaefer*
Computer Science Department, AGH University of Science and Technology
Kraków, Poland

Maciej Smołka†
Institute of Computer Science, Jagiellonian University
Kraków, Polnad
{*grochows,smolka,schaefer*}*@ii.uj.edu.pl*

Abstract. The paper introduces the formal description of a computing multi-agent system (MAS), its architecture and dynamics (sections 2–4). The optimal scheduling problem for the MAS as well as a way of its verification are presented in terms of such a model (section 5). A brief report of test results published previously in [13, 3, 4, 8] is contained in the section 6.

Keywords: Distributed computations, multi-agent systems, task scheduling

1. Motivation

The application of the MAS paradigms for designing and implementing large-scale distributed computing system is rather unusual. Typical solutions in this area are based on the low level communication libraries (PVM, MPI). Even recent ideas arising in the cross grid installations (Condor [12], Globus OGSA [2]) do not include the active software modules for a task implementation. We may expect a reasonable improvement in the phase of design, implementation and running if a distributed application is a variable-size collection of intelligent agents. First of all we can speed up the design of computational modules by

*Address for correspondence: Computer Science Department, AGH University of Science and Technology, Al.Mickiewicza 30, 30-059 Kraków, Poland
†This author has been supported by the State Committee for Scientific Research of the Republic of Poland under research grants 2 P03A 003 25 and 7 T07A 027 26

wrapping the computational tasks in the agent shell, which is responsible for some management duties (task scheduling, partitioning, etc.). Basing on the idea of task diffusion (see e.g. [7]) we propose a scheduling policy which consists of an on-demand task partitioning and a task remapping obtained by the dynamic agent creation and the agent migration within a computer network. A simple diffusion rule is applied locally and simultaneously for groups of agents. It decreases significantly the computational and communication costs arising in centralized scheduling policies. The paper introduces the formal description of the MAS under consideration, the underlying agent architecture and the dynamics of the whole system (sections 2–4). The optimal scheduling problem for a MAS as well as a way of its verification is presented in terms of a such model (section 4). A brief report of test results published in [13, 3, 4, 8] is contained in section 6.

2. The Architecture

2.1. Formal description

The MAS under consideration that allows the diffusion governed scheduling is a collection of: *a computational environment* (MAS platform) and *a computing application* composed of mobile agents called *Smart Solid Agents* (SSA). The computational environment is a quadruple $(\mathbf{N}, B_H, perf, conn)$, where:

$\mathbf{N} = \{P_1, \ldots, P_n\}$, where P_i is a Virtual Computation Node (VCN). Each VCN can maintain more than one agent (the number of hardware processors used is not relevant in our assumptions).

B_H is the connection topology $B_H = \{N_1, \ldots, N_n\}, N_i \subset \mathbf{N}$ is an immediate neighborhood of P_i (including P_i as well).

$perf = \{perf_1, \ldots, perf_n\}, perf_i : \mathbb{R}_+ \to \mathbb{R}_+$ is a family of functions, which describes relative performance of all VCN with respect to the total memory request M^i_{total} of all allocated agents. If M^i_{total} on P_i is small, $perf_i$ turns back the constant value, which depends only on the CPU architecture. As M^i_{total} grows larger, $perf_i$ decreases due to the intensive memory swap utilization.

$conn : \mathbf{N} \times \mathbf{N} \to \mathbb{R}_+$ is a function, which describes up-to-date connection speed between two VCNs. The connection speed is expressed in number of bytes per second which can be transmitted as a body of regular platform messages.

Each SSA is represented by the pair $A_i = (T_i, S_i)$ where T_i is the computational task executed by agent, including all data required for computation, and S_i stands for the shell responsible for the agent's logic. The index i stands for an unambiguous agent identifier.
Each task T_i has to denominate the current requirement for computational power (E_i, M_i) where: E_i is the task remaining time measured in units common for all application tasks, and M_i is the RAM requirement in bytes. Task T_i can also provide information about its future communication with other tasks described by the set of pairs $C_i = \{(T_\zeta, data_\zeta)\}$ where T_ζ is another application task, and $data_\zeta$ is the amount of data, expressed in bytes, which will be exchanged between tasks T_i and T_ζ.
Another important condition we impose on the task is that it must allow pausing and continuation of its computations. The pausing is needed for the task hibernation in the case of the agent migration or partitioning, and the continuation is needed to restore the paused job. In particular the task can be designed in such a way that it can do its job between checkpoints and during the checkpoint operation it

saves its current state. Moreover each task T_i can be partitioned into two subtasks $T_i \rightarrow \{T_{i_1}, T_{i_2}\}$ such that $E_i > E_{i_j}$, $M_i > M_{i_j}$, $j = 1, 2$. The task partitioning rule depends strongly on the computational problem to be solved (see [9]).

2.2. The state of the computing application

The state of *the computing application* is the triple $(\mathbf{A}_t, G_t, Sch_t)$, $t \in [0, +\infty)$ where:

$\mathbf{A}_t$ is the set of application agents, $\mathbf{A}_t = \{A_{\xi_j}\}_{\xi_j \in I_t}$, I_t is the set of indices of agents active at the time t,

G_t is the tree representing agents partitioning at the time t. All agents constitute the set of nodes $\bigcup_{\xi \in \Theta} A_\xi$, $\Theta = \bigcup_{j=0}^{t} I_j$, while edges of G_t show the partitioning history. All information on how to rebuild G_t is spread among all agents in such a way that each of them is aware of only its neighbors in the tree.

$\{Sch_t\}_{t \in [0,+\infty)}$ is the family of functions such that $Sch_t : \mathbf{A}_t \rightarrow \mathbf{N}$ is the current schedule of application agents among the MAS platform servers. The function is represented by the sets ω_j of agents' indices allocated on each $P_j \in \mathbf{N}$. Each of ω_j is locally stored and managed by P_j.

The shell S_i communicates with both T_i and the local server $P_j = Sch(A_i)$. It supports inter-task communication and queries task requirements for resources, implementing the necessary logic to perform scheduling as well. Each server $P_j \in \mathbf{N}$ periodically asks all local agents (allocated on P_j) for their requirements and computes the local load concentration

$$L_j = \frac{E^j_{total}}{perf_j(M^j_{total})} \quad \text{where} \quad E^j_{total} = \sum_{i \in \omega_j} E_i \quad \text{and} \quad M^j_{total} = \sum_{i \in \omega_j} M_i \tag{1}$$

Then P_j communicates with neighboring servers and establishes

$$\mathbf{L}_j = \{(L_\zeta, E^\zeta_{total}, M^\zeta_{total}, perf_\zeta)\} \quad \text{where } \zeta \text{ is such that } P_\zeta \in N_j \tag{2}$$

as well as the set of node indices Q_j such that

$$k \in Q_j \iff k \neq j, P_k \in N_j, L_j - L_k > 0 \tag{3}$$

The current values of both $\mathbf{L}_j$ and Q_j are available to the local agents.
Moreover we may include inter-agent communication to our model, which is based on the set C_i containing information about the future communication activity of the task T_i. The communication cost associated with agent A_i containing task T_i is evaluated as

$$c_i = \sum_{(T_\zeta, data_\zeta) \in C_i} \frac{data_\zeta}{conn(Sch(A_i), Sch(A_\zeta))} \tag{4}$$

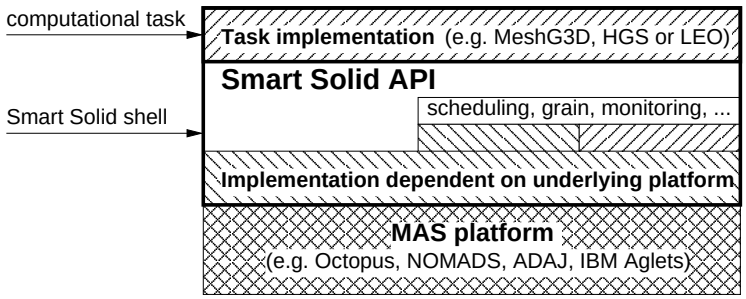

Figure 1. Architecture of Smart Solid Agent, located on the top of MAS platform (for descriptions of MAS platforms see [11, 1, 5]).

2.3. Three layer architecture

Our solution is structured as the three layer architecture due to various reasons. The most important is the ability to separate responsibilities for several logically independent aims such as solving numerical task, finding needed resources together with balancing the load of entire computing system and supporting hardware and operating systems by a MAS platform. Such a separation allows to do an independent research in different areas and combine research results into one complex system, with the possibility of exchanging the implementation of various architectural elements.
Figure 1 depicts dependencies between Smart Solid Agent shell, implementation of computational task and the part of the agent implementation dependent on the underlying platform. White rectangles represent API's (Application Programming Interfaces) of Smart Solid Agent including interfaces for scheduling, grain controlling, monitoring, whilst shaded rectangles represent relevant implementations which can be extended or exchanged to achieve particular needs (see section 6). Smart Solid provides default implementations of these interfaces where the most important is the implementation of the scheduling strategy realized as an analogy to diffusion phenomena.

3. Diffusion of the Smart Solid agent

We introduce the *binding energy* $\mathbf{E}_{i,j}$ of the agent A_i allocated on VCN P_j characterized by the following conditions:

$$\mathbf{E}_{i,j} \text{ is a descending function of } (E_i, c_i) \text{ and a nonascending function of } L_j \tag{5}$$

One of the simplest form of the binding energy utilized in computational tests (see section 6) is

$$\mathbf{E}_{ij} = \max\{E_{min}, (\alpha_1 E_i + \alpha_2 c_i + \alpha_3 L_j)^{-1}\} \tag{6}$$

where $\alpha_1, \alpha_2, \alpha_3$ are the proper scaling parameters and E_{min} stands for the minimum binding energy assigned to each agent.
We assume that the agent A_i is able to evaluate dynamically its binding energy for other nodes from the neighborhood N_j using the information contained in $\mathbf{L}_j$ as well as its communication cost $\{c_\zeta\}, \zeta \in N_j$.

The current value of the binding energy gradient is a vector defined by

$$\nabla_{i,j}^{t} = ((j,l), \mathbf{E}_{i,l} - \mathbf{E}_{i,j}) \tag{7}$$

where $P_j = Sch(A_i)$ and $l \in Q_j$ are such that $\mathbf{E}_{i,l} - \mathbf{E}_{i,j} = \max_{\zeta \in Q_j}\{\mathbf{E}_{i,\zeta} - \mathbf{E}_{i,j}\}$.
An agent A_i allocated on P_j migrates to P_l indicated by $\nabla_{i,j}^{t}$ if the binding energy $\mathbf{E}_{i,l}$ on the destination VCN exceeds the current $\mathbf{E}_{i,j}$ more than ϵ. The threshold ϵ stands for the migration parameter.
In general Smart Solid Agent $A_i = (T_i, S_i)$ currently allocated on $P_j \in \mathbf{N}$ can perform the following actions:

(a-1) Execute task T_i (solve and communicate with other agents).

(a-2) Pause T_i .

(a-3) Continue T_i .

(a-4) Denominate own load requirements (E_i, M_i) and future communication description C_i.

(a-5) Compute $\nabla_{i,j}^{t}$ and check the condition $\mathbf{E}_{i,l} - \mathbf{E}_{i,j} > \epsilon$.

(a-6) Partition $T_i \rightarrow \{T_{i_1}, T_{i_2}\}$, create child agents $\{A_{i_j} = (T_{i_j}, S_{i_j})\}, j = 1, 2$.

(a-7) Migrate to $P_l \in \mathbf{N}, l \neq j$.

(a-8) Disappear.

These actions allow A_i to accomplish two goals:

(G-1) Perform computation of carried task by executing action (a-1) and then perform action (a-8) when the task is done.

(G-2) Find a better execution environment. We suggest the following algorithm utilizing actions (a-2) - (a-8).

if $Q_j = \emptyset$ **then**
  continue T_i
else
  compute $\nabla_{i,j}^{t}$
  if $\mathbf{E}_{i,l} - \mathbf{E}_{i,j} > \epsilon$ **then**
    pause T_i; migrate along the gradient $\nabla_{i,j}^{t}$; continue T_i
  else
    partition $T_i \rightarrow \{T_{i_1}, T_{i_2}\}$;
    create $\{A_{i_j} = (T_{i_j}, S_{i_j})\}, j = 1, 2$; $\{G_t$ gets modified$\}$
    disappear
  end if
end if

The overall SSA intention is to accomplish the goal (G-1) in the shortest possible time. If the agent recognizes the local VCN resources as insufficient, it tries to accomplish the goal (G-2). On the other hand, P_j may force $\{A_i\}, i \in \omega_j$ to realize goal (G-2) when its performance is endangered.

4. The MAS dynamics

The diffusion-based scheduling strategy has proven to be simple and efficient (cf. section 6) but it lacks a theoretical background allowing to determine if it is optimal or quasi-optimal in any sense. In this section we state a formal mathematic model for multi-agent computations. This model was presented first in [10], here we give its version with slightly changed notation and a little bit more explanation on introduced quantities. Consider for a while that we are given a space of all possible agents and denote it by $\mathbf{A}$. We shall consider discrete-time evolution of a given MAS. Let us introduce the notion of the *vector weight of an agent* which is the mapping

$$w : \mathbb{N} \times \mathbf{A} \longrightarrow \mathbb{R}_+^2$$

whose components are E_i and M_i as introduced earlier. Assume that we know how the total weight of child agents after partition depends on their parent's weight before partition and that this dependency is componentwise, i.e. we know the functions $f^1, f^2 : \mathbb{R}_+ \to \mathbb{R}_+$ such that in the case of partition $A \to \{A_1, A_2\}$ we have

$$w_{t+1}^i(A_1) + w_{t+1}^i(A_2) = f^i(w_t^i(A))$$

for $i = 1, 2$. Such an assumption seems realistic, in simple cases f^i can be even the identity. Note that probably the weakest reasonable assumption on f^i is that

$$f^i(0) = 0. \tag{8}$$

Next denote by

$$W : \mathbb{N} \times \mathbf{N} \longrightarrow \mathbb{R}_+$$

the *total weight of all agents allocated on a virtual node at any time*, i.e.

$$W_t(P) = \sum_{Sch_t(A)=P} w_t(A)$$

(obviously we put 0 if no agent is maintained by P). The main idea of introducing such a notion is the need to find a global quantity describing the state of the system appropriately and allowing us to avoid considering the dynamics of a single agent.
In the sequel we shall assume that the number of virtual nodes

$$\sharp\mathbf{N} = N$$

is *fixed*. Thus we can consider W_t as a nonnegative vector in $\mathbb{R}^{2N}$ whose j-th component corresponds to E_{total}^j and $(N + j)$-th component corresponds to M_{total}^j. In fact we shall treat W_t as a *stochastic (vector-valued) process*. Now we shall state the equations of evolution of W_t (i.e. *state equations* of our system). To this end consider three different cases.

1. *'Established' evolution*, i.e. the one without migrations or partitions. Then the state equation has the form

   $$W_{t+1} = F_t(W_t)$$

   where F_t is a given stochastic nonnegative vector field.

2. *Partition at node j.* Then we have

$$\begin{cases} W_{t+1}^{j} & = (1-\alpha_j^E(W_t))W_t^j + f^1(\alpha_j^E(W_t)W_t^j) \\ W_{t+1}^{N+j} & = (1-\alpha_j^M(W_t))W_t^{N+j} + f^2(\alpha_j^M(W_t)W_t^{N+j}) \\ W_{t+1}^{i} & = W_t^i \qquad \text{for } i \neq j, N+j \end{cases}$$

where $\alpha_j^E, \alpha_j^M : \mathbb{R}_+^N \to [0,1]$ are the proportions of the weight components of splitting agents to the corresponding components of the total weight of all agents at node j.

3. *Migration from j to k.* In this case the state equations have the form

$$\begin{cases} W_{t+1}^{j} & = (1-\beta_{jk}^E(W_t))W_t^j \\ W_{t+1}^{N+j} & = (1-\beta_{jk}^M(W_t))W_t^{N+j} \\ W_{t+1}^{k} & = W_t^k + \beta_{jk}^E(W_t)W_t^j \\ W_{t+1}^{N+k} & = W_t^{N+k} + \beta_{jk}^M(W_t)W_t^{N+j} \\ W_{t+1}^{i} & = W_t^i \qquad \text{for } i \neq j, k, N+j, N+k \end{cases}$$

with $\beta_{jk}^E, \beta_{jk}^M$ analogous to α^E, α^M.

Putting all the above cases together we obtain the following state equations

$$\begin{cases} W_{t+1}^{i} & = F_t^i(\tilde{W}_t) + f^1\left(\alpha_{i,t}^E(W_t)W_t^i\right) - \alpha_{i,t}^E(W_t)W_t^i \\ & \quad - \sum_{k\neq i}\beta_{ik,t}^E(W_t)W_t^i + \sum_{j\neq i}\beta_{ji,t}^E(W_t)W_t^j \\ W_{t+1}^{N+i} & = F_t^{N+i}(\tilde{W}_t) + f^2\left(\alpha_i^M(W_t)W_t^{N+i}\right) - \alpha_i^M(W_t)W_t^{N+i} \\ & \quad - \sum_{k\neq i}\beta_{ik}^M(W_t)W_t^{N+i} + \sum_{j\neq i}\beta_{ji}^M(W_t)W_t^{N+j} \end{cases} \tag{9}$$

where

$$\begin{cases} \tilde{W}_t^i & = \left(1-\alpha_{i,t}^E(W_t) - \sum_{k\neq i}\beta_{ik,t}^E(W_t)\right)W_t^i \\ \tilde{W}_t^{N+i} & = \left(1-\alpha_{i,t}^M(W_t) - \sum_{k\neq i}\beta_{ik,t}^M(W_t)\right)W_t^{N+i} \end{cases}$$

for $i = 1, \ldots, N$. To simplify the notation put

$$u_{ii,t}^E = \alpha_{i,t}^E,\ u_{ii,t}^M = \alpha_{i,t}^M \tag{10}$$

and

$$u_{ij,t}^E = \beta_{ij,t}^E,\ u_{ij,t}^M = \beta_{ij,t}^M \text{ for } i \neq j. \tag{11}$$

Then the state equations have the simpler form

$$\begin{cases} W_{t+1}^{i} & = F_t^i(\tilde{W}_t) + f^1\left(u_{ii,t}^E(W_t)W_t^i\right) - \left(\sum_{k=1}^N u_{ik,t}^E(W_t)\right)W_t^i \\ & \quad + \sum_{j\neq i}u_{ji,t}^E(W_t)W_t^j \\ W_{t+1}^{N+i} & = F_t^{N+i}(\tilde{W}_t) + f^2\left(u_{ii,t}^M(W_t)W_t^{N+i}\right) - \left(\sum_{k=1}^N u_{ik,t}^M(W_t)\right)W_t^{N+i} \\ & \quad + \sum_{j\neq i}u_{ji,t}^M(W_t)W_t^{N+j} \end{cases} \tag{12}$$

with

$$\begin{cases} \tilde{W}_t^i & = \left(1 - \sum_{k=1}^N u_{ik,t}^E(W_t)\right) W_t^i \\ \tilde{W}_t^{N+i} & = \left(1 - \sum_{k=1}^N u_{ik,t}^M(W_t)\right) W_t^{N+i} \end{cases}$$

Note that in the above equations we have the *control* term

$$u = (u^E, u^M) \tag{13}$$

where

$$u^E, u^M : \mathbb{N} \times \mathbb{R}_+^{2N} \longrightarrow [0,1]^{N \times N}. \tag{14}$$

From the nature of our problem it follows that without loss of generality we can impose the following conditions on all control strategies

$$\begin{cases} u_{ij,t}^E(x) \cdot u_{ji,t}^E(x) = 0, \\ u_{ij,t}^M(x) \cdot u_{ji,t}^M(x) = 0 \text{ for } i \neq j, \\ \sum_{k=1}^N u_{ik,t}^E(x) \leq 1, \\ \sum_{k=1}^N u_{ik,t}^M(x) \leq 1 \text{ for } i = 1, \ldots, N. \end{cases} \tag{15}$$

for any $t \in \mathbb{N}$ and $x \in \mathbb{R}_+^{2N}$. The first pair of equalities means that at a given time migrations between two nodes may happen in only one direction. The remaining conditions say that the number of agents leaving a node must not exceed the number of agents present at the node just before the migration.
The initial state of equation (12)

$$W_0 = \hat{W} \tag{16}$$

often has all but one components equal to 0 which means that an application starts with one big agent or a batch of agents allocated on a single virtual node but this is not the most general case and we need not make such an assumption.

5. The optimal scheduling problem

In real computational environments both the available memory and the maximal computation time is limited and naturally quantized so we can safely assume that W_t has finite number of states. Hence for the following considerations we assume that it is a controlled Markov chain with a finite state space $\{v_0 = 0, v_1, \ldots, v_M\}$. We also assume that 0 is its absorbing state (cf. [6]). This condition is very natural in our situation since 0 is the desired final state of our system. Reaching 0 means that the computations are done so we do not want the system to leave this state.
Consider two examples of cost functionals which seem appropriate for multi-agent computations. The first one is the expected total time of computations

$$V_T(u; \hat{W}) = E\left[\inf\{t \geq 0 : W_t = 0\}\right]. \tag{17}$$

The second takes into account the mean load balancing over time. It has the following form

$$V_L(u; \hat{W}) = E\left[\sum_{t=0}^{\infty} \sum_{i=1}^{N} (L_t^i - \overline{L}_t)^2\right] \tag{18}$$

where

$$L_t^i = \frac{W_t^i}{perf_i(W_t^{N+i})}$$

is the load concentration and

$$\overline{L}_t = \frac{1}{N}\sum_{i=1}^{N} L_t^i$$

is its mean over all nodes. Let us introduce the following notation for the set of admissible controls

$$\mathbf{U} = \{u = (u^E, u^M) : u \text{ satisfies conditions (14) and (15)}\}. \tag{19}$$

Assume that V has either the form (17) or (18). Given an initial configuration $\hat{W}$ our *optimal scheduling problem* is now to find such control strategy $u^* \in \mathbf{U}$ that

$$V(u^*; \hat{W}) = \min\{V(u; \hat{W}) : \ u \in \mathbf{U}, \ W \text{ is a solution of (12), (16)}\}. \tag{20}$$

Now let us consider the existence of optimal strategies. To this end let us denote by $\overline{F}_t(W_t, u_t(W_t))$ the right hand side of (12) and by

$$p_{ij}(t, \hat{u}) = P(\overline{F}_t(v_i, \hat{u}) = v_j) \tag{21}$$

the transition probabilities of W_t. Moreover denote by $\mathbb{U}$ the set of all possible values of controls evaluated at any state and any time, i.e. the set of those elements of $[0,1]^{2(N\times N)}$ which satisfy conditions (15). Finally denote by

$$R(\hat{u}) = [p_{ij}(0, \hat{u}(v_i)]_{i=1}^N$$

the 'probably not absorbing' part of the transition matrix for $\hat{u} \in \mathbb{U}$ and by $R^n(\hat{u})$ the analogous part of n-step transition matrix obtained by applying the stationary control $u_t = \hat{u}$.

Proposition 5.1. Assume that

1. Functions $p_{ij}(t, \cdot)$ are continuous on $\mathbb{U}$
2. For $V = V_T$ matrix $R^n(\hat{u})$ is a contraction for some $\hat{u} \in \mathbb{U}, n \geq 1$.
3. For $V = V_L$ matrix $R^M(\hat{u})$ is a contraction for every $\hat{u} \in \mathbb{U}$.

Then there exists an optimal solution of (20).

Proof:
Notice that $\mathbb{U}$ is a compact set and that both cost functionals do not depend explicitly on u. Then it is a straightforward consequence of [6, Theorem 4.2] provided that we notice moreover that when $V = V_T$ we have

$$V_T(u, \hat{W}) = E\left[\sum_{t=0}^{\infty} k(W_t)\right] + 1$$

where

$$k(v_i) = 1$$

for all $v_i \neq 0$ and $k(0) = 0$. □

6. Numerical tests

This section presents the performed experiments and points out on various efficiency aspects of the computing MAS application. The results cover two major problem domains examined in the course of our experiments: 3D Mesh Generator and Hierarchic Genetic Strategy (HGS). Here we present their short description with follow-up references.

- The 3D Mesh Generator [4] is a CAD/CAE task implementation. Each agent is equipped with a single part of a partitioned solid, for which the mesh has to be generated. If the solid is partitioned in a fixed way, the agents' number and size is also fixed and so are their computation power demands.

- The HGS [14] is a stochastic, hierarchical genetic algorithm optimizing a given function on a defined domain. The application produces agents dynamically during the runtime. A single HGS agent is a container, which starts executing its internal populations as soon as it finds a suitable computation environment. The total number of agents at the execution time may be different even for the same input data. On the other hand, the sizes of agents are almost equal.

The major aim of the 3D Mesh Generator application was to prove the efficiency of the diffusion-based scheduling in the case of a regular problem. The main points of the results' analysis was to check if the local, diffusion-based scheduling policy does not put too much overhead on the total execution time and if the available resources were fully utilized.
The paper [4] presents the runtime properties of the 3D Mesh Generator application in detail as well as the actual realization of the analytical model. It concludes that the diffusion-based, local scheduling policy is well suited for such regular problems — available resources were utilized at 96%. The results also confirm that the communication factor may be omitted for a certain class of applications since the total communication overhead is minimal. Also the communication required for the diffusion (examining a VCN node neighborhood and reevaluating the binding energy for every agent) did not decrease the system efficiency.

The HGS application [7] is an example of irregular, stochastic application, where agents are created dynamically as the genetic populations examine the optimized function domain thoroughly. On the contrary to the 3D Mesh Generation single agent acts as a *container* for the processed genetic populations, which are treated as computational tasks executed sequentially.
The communication overhead and dynamic scheduling efficiency were tested within this experiment. The effectiveness of the dynamic scheduling was tested versus centralized, greedy scheduling policy (a Round Robin solution) utilizing low level network message passing mechanisms (Java RMI). The final conclusion can be stated that the diffusion-based local scheduling performs well also in the case of an irregular stochastic problem with variable number of agents. Although it is moderately slower for small problems (about 3 times slower) than the dedicated fast low-level solution, it proves very adaptive and its local scheduling evaluation does not significantly decrease the solution's effectiveness — the observed communication overhead is about 5% of total execution time.

Serial time [sec]	Diffusive Scheduling			Round-Robin	
	Total number of agents	Parallel time [sec]	Speedup	Parallel time [sec]	Speedup
4580	218	304	15,06579	209	21,91388
6891	244	371	18,57412	318	21,66981
6766	299	299	22,62876	334	20,25749
3387	122	374	9,05615	163	20,77914
4267	235	294	14,51361	206	20,71359
2687	94	228	11,78509	131	20,51145
4221,9	**183,7**	**288,2**	**14,44104**	**204,9**	**20,66099**

Table 1. Speedup of HGS computations for the Griewangk objective with Round-Robin and Diffusive Scheduling. Arbitrary selected times for most significant experiments and mean values computed for all conducted experiments (last row) are shown.

7. Conclusions and Further Research

The diffusion analogy as well as the MAS technology give way effectively to designing a local diffusion-based scheduling strategy for a distributed environment. Its effectiveness is achieved by the low complexity of local scheduling rules and the lack of intensive communication required by centralized schedulers. However when we want to find out if our heuristic strategy is optimal (or probably ε-optimal) in any strict mathematical sense, we need to provide an appropriate mathematical model which would describe the considered system. Such a model is presented in sections 4 and 5 together with a theoretical result on the existence of optimal scheduling strategies. From the model we expect many further practical results, including the most interesting ones — optimality conditions. There is also a set of related problems to consider, e.g. one needs to estimate (and compute) the quantities constituting equations (12).
As mentioned in section 6, a series of various experiments have been passed (see [13, 3, 4, 8]). They showed the efficiency of the diffusion governed scheduling, together with the easiness of the MAS driven approach to distributed computations' deployment and maintenance.
Extending the current solution we intend to build OGSA compatible GRID services [2] to expose MAS platforms to the Internet and allow migration of Smart Solid Agents between many local computer networks.

References

[1] Felea, V., Olejnik, R., Toursel, B.: ADAJ: A Java Distributed Environment for Easy Programming Design and Efficient Execution, *Schedae Informaticae*, **13**, 2004, 9–36.

[2] The Globus Alliance, http://www.globus.org/: *Open Grid Services Architecture*.

[3] Grochowski, M., Schaefer, R., Uhruski, P.: An Agent-Based Approach to a Hard Computing System — Smart Solid, *Proceedings of the International Conference on Parallel Computing in Electrical Engineering (PARELEC 2002)*, IEEE Computer Society Press, Warsaw, Poland, 22–25 September 2002.

[4] Grochowski, M., Schaefer, R., Uhruski, P.: Diffusion Based Scheduling in the Agent-Oriented Computing Systems, *Lecture Notes in Computer Science*, **3019**, 2004, 97–104.

[5] IBM, http://aglets.sourceforge.net/: *IBM Aglets*.

[6] Kushner, H.: *Introduction to Stochastic Control*, Holt, Rinehart and Winston, 1971.

[7] Luque, E., Ripoll, A., Cortés, A., Margalef, T.: A Distributed Diffusion Method for Dynamic Load Balancing on Parallel Computers, *Proceedings of EUROMICRO Workshop on Parallel and Distributed Processing*, IEEE Computer Society Press, San Remo, Italy, January 1995.

[8] Momot, J., Kosacki, K., Grochowski, M., Uhruski, P., Schaefer, R.: Multi-Agent System for Irregular Parallel Genetic Computations, *Lecture Notes in Computer Science*, **3038**, 2004, 623–630.

[9] Schaefer, R., Flasiński, M., Toporkiewicz, W.: Optimal Stochastic Scaling of CAE Parallel Computations, *Lecture Notes in Computer Science*, **1424**, 1998, 557–564.

[10] Smołka, M., Grochowski, M., Uhruski, P., Schaefer, R.: The Dynamics of Computing Agent Systems, Lecture Notes in Computer Science, **3516**, 2005, 727–734.

[11] Suri, N., Bradshaw, J. M., Breedy, M. R., Groth, P. T., Hill, G. A., Jeffers, R., Mitrovich, T. S.: An Overview of the NOMADS Mobile Agent System, *Proceedings of the European Conference on Object-Oriented Programming (ECOOP'2000)*, Sophia Antipolis and Cannes, France, 2000.

[12] Thain, D., Tannenbaum, T., Livny, M.: Condor and the Grid, in: *Grid Computing: Making the Global Infrastructure a Reality* (F. Berman, G. Fox, T. Hey, Eds.), Wiley, 2002.

[13] Uhruski, P., Grochowski, M., Schaefer, R.: Multi-Agent Computing System in a Heterogeneous Network, *Proceedings of the International Conference on Parallel Computing in Electrical Engineering (PARELEC 2002)*, IEEE Computer Society Press, Warsaw, Poland, 22–25 September 2002.

[14] Wierzba, B., Semczuk, A., Kołodziej, J., Schaefer, R.: Hierarchical Genetic Strategy with Real Number Encoding, *Proceedings of 6th Conference on Evolutionary Algorithms and Global Optimization*, Wydawnictwa Politechniki Warszawskiej, Łagów Lubuski, Poland, 2003.