The Dynamics of Computing Agent Systems

Smołka M.*, Uhruski P., Schaefer R., and Grochowski M.

Institute of Computer Science, Jagiellonian University, Kraków, Poland
{smolka, uhruski, schaefer, grochows}@ii.uj.edu.pl

Abstract. The paper presents the Multi Agent System (MAS) designed for the large scale parallel computations. The special kind of diffusion-based scheduling enables to decompose and allocate the migrable computing agents basing only of the local information. The paper introduces the formal model of the MAS under consideration in order to depict the roles of agent behavior and the whole system dynamics. The optimal scheduling problem for MAS as well as the way of its verification was presented in terms of such model. The brief report of the test results is stressed in the section 6.

1 Motivation

The Multi-Agent System (MAS) governed by the local diffusion scheduling is a reasonable alternative for the centrally governed distributed computing systems. Although the idea of mobile task diffusion is well known since more then ten years (see. e.g. [6]) we propose the policy that consists in on-demand task partitioning and task remaping obtained by the dynamic agent creation and migration. It was implemented and initially tested for the regularly and irregularly concurrent computations (see [2, 1, 5, 7]). The brief report of the test results is stressed in the section 6. The paper introduces the formal model of the MAS under consideration in order to depict the roles of agent behavior and the whole system dynamics. The optimal scheduling problem for MAS as well as the way of its verification was presented in terms of such model.

2 Formal Description of the Architecture

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 triple $(\mathbf{N}, B_H, perf)$, where:

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

* 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.

V.S. Sunderam et al. (Eds.): ICCS 2005, LNCS 3516, pp. 727–734, 2005.

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. If M^i_{total} is larger, the $perf_i$ decreases due to the intensive swap utilization.

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. Another important condition we imposed for the task is that it must allow pausing and continuation of it's computation. Pausing is needed for the hibernating task in case of agent migration or partitioning, and continuation is needed to restore the paused job. In particular it can be designed in such a way that it can work from one checkpoint to the next one, and during this checkpoint operation, it saves its present state. Moreover each task T_i can be partitioned into two subtasks $T_i \to \{T_{i_1}, T_{i_2}\}$ 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 [3]).
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 G_t edges show the partitioning history. All information on how to rebuilt G_t is spread among all agents such that each of them knows only its neighbors in the tree.

$\{Sch_t\}_{t \in [0,+\infty)}$ is the family of functions such that $Sch_t : \mathbf{A}_t \to \mathbf{N}$ 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 as well as implements the necessary logic to perform scheduling. 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

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

Then P_j communicates with neighboring servers and establishes

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

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.

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: $\mathbf{E}_{i,j}$ is a descending function of E_i and a nonascending function of L_j. One of the simplest form of the binding energy utilized in computational tests (see section 6) is $\mathbf{E}_{ij} = max\{E_{min}, (\alpha_1 E_i + \alpha_2 L_j)\}$ where α_1, α_2 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 may dynamically evaluate its binding energy for other nodes from the neighborhood N_j using the information contained in $\mathbf{L}_j$. The current value of the binding energy gradient is a vector defined by:

$$\nabla^t_{i,j} = ((j,l), \mathbf{E}_{i,l} - \mathbf{E}_{i,j}) \text{ where } P_j = Sch(A_i) \text{ and } l \in Q_j \text{ is such that } \mathbf{E}_{i,l} - \mathbf{E}_{i,j} = \max_{\zeta \in Q_j}\{\mathbf{E}_{i,\zeta} - \mathbf{E}_{i,j}\} \tag{4}$$

An agent A_i allocated on P_j migrates to P_l indicated by $\nabla^t_{i,j}$ 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).
(a-5) Compute $\nabla^t_{i,j}$ 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 following the algorithm utilizing actions (a-2) - (a-8).

If $Q_j = \emptyset$ **then** continue T_i
  else { compute $\nabla^t_{i,j}$;
    If $\mathbf{E}_{i,l} - \mathbf{E}_{i,j} > \epsilon$
      then { pause T_i; migrate along the gradient $\nabla^t_{i,j}$; continue T_i }
      else { Partition $T_i \to \{T_{i_1}, T_{i_2}\}$;
        create $\{A_{i_j} = (T_{i_j}, S_{i_j})\}, j = 1, 2$; // G_t gets modified
        disappear }
    }.

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 shall try to state a formal mathematic model for multi-agent computations. 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

$$w^i_{t+1}(A_1) + w^i_{t+1}(A_2) = f^i(w^i_t(A))$$

for $i = 1, 2$. Such an assumption seems realistic, in simple cases f^i can be even the identity.

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^j_{total} and $(N + j)$-th component corresponds to

M^j_{total}. 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). Let F_t be a time-dependent stochastic nonnegative vector field on $\mathbb{R}^{2N}_+$ describing the dynamics of our system in 'established' state, i.e. when there are neither migrations nor partitions. Let $u^E_{ij,t}(W_t), u^M_{ij,t}(W_t) \in [0,1]$ denote the proportions of the weight components of agents migrating from node i to node j to the corresponding components of the total weight of all agents at node i and let $u^E_{ii,t}(W_t), u^M_{ii,t}(W_t)$ denote the proportions of the weight components of splitting agents to the corresponding components of the total weight of all agents at node i. Then the state equations have the form:

$$\begin{cases} W^i_{t+1} = F^i_t\left(\left[(1-\sum_{k=1}^N u^E_{ik,t}(W_t))W^i_t\right]_{i=1}^N\right) \\ \qquad +f^1\left(u^E_{ii,t}(W_t)W^i_t\right) - (\sum_{k=1}^N u^E_{ik,t}(W_t))W^i_t \\ \qquad +\sum_{j\neq i} u^E_{ji,t}(W_t)W^j_t \\ W^{N+i}_{t+1} = F^{N+i}_t\left(\left[(1-\sum_{k=1}^N u^M_{ik,t}(W_t))W^{N+i}_t\right]_{i=1}^N\right) \\ \qquad +f^2\left(u^M_{ii,t}(W_t)W^{N+i}_t\right) - (\sum_{k=1}^N u^M_{ik,t}(W_t))W^{N+i}_t \\ \qquad +\sum_{j\neq i} u^M_{ji,t}(W_t)W^{N+j}_t \end{cases} \tag{5}$$

for $i = 1, \ldots, N$. Note that $u = (u^E, u^M)$ where $u^E, u^M : \mathbb{N} \times \mathbb{R}^N_+ \longrightarrow [0,1]^{N\times N}$ is in fact a *control* of our system. From the nature of our problem it is easy to see that any control strategy must satisfy the following conditions for any $t \in \mathbb{N}$ and $x \in \mathbb{R}^N_+$

$$\begin{cases} u^E_{ij,t}(x)\cdot u^E_{ji,t}(x) = 0, u^M_{ij,t}(x)\cdot u^M_{ji,t}(x) = 0 \text{ for } i \neq j, \\ \sum_{k=1}^N u^E_{ik,t}(x) \leq 1, \sum_{k=1}^N u^M_{ik,t}(x) \leq 1 \qquad \text{for } i = 1, \ldots, N. \end{cases} \tag{6}$$

The first pair of equalities means that at a given time migrations between two nodes may happen in only one direction. The remaining conditions mean 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 (5)

$$W_0 = \hat{W} \tag{7}$$

usually has all except one components equal to 0 which means that a system starts with one agent or a batch of agents placed on a single VCN but of course in general we need not make such an assumption.

5 The Optimal Scheduling Problem

Now let us propose two examples of cost functionals which seem appropriate for multi-agent computations. The first is the expected total time of computations

$$V(u;\hat{W}) = E(\min\{t \geq 0 : \sum_{i=1}^N W^i_t = 0\}), \tag{8}$$

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

$$V(u;\hat{W}) = E(\sum_{t=0}^{\infty}\sum_{i=1}^{N}(L_t^i - \overline{L}_t)^2) \tag{9}$$

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

$$\mathbf{U} = \{u = (u^E, u^M) : \mathbb{N} \times \mathbb{R}_+^N \to [0,1]^{2(N\times N)} |\ u \text{ satisfies conditions (6)}\}.$$

Assume that V has either the form (8) or (9). Given an initial configuration $\hat{W}$ our *optimal scheduling problem* is now to find such control strategy $u^* \in \mathbf{U}$ that

$$V(u^*;\hat{W}) = \min\{V(u;\hat{W}) :\ u \in \mathbf{U},\ W \text{ is a solution of } (5),(7)\}. \tag{10}$$

6 Numerical Tests

This section presents performed experiments and discusses how they fit presented analytical model. The results cover two major problem domains examined in course of our experiments: Mesh Generator (MG) and Hierarchic Genetic Strategy (HGS). The presented tests are broadly described in [5, 7] while papers [1, 2] contain the MAS implementation details. Computations were run in the same physical environment with LAN of 30 to 50 PCs connected with 100MBit network.

6.1 Mesh Generator

The MG is a CAD/CAE task implementation. Each agent is equipped with a single part of the partitioned solid, for which the mesh has to be generated. The application had the following execution path:

1. Create all the agents, equip each with the part of the solid to generate the mesh for. The agent's size varies from very small to large depending on the solid partitioning and each piece shape - the more complex the shape is, the bigger mesh is generated.
2. Agents are put on the MAS network and distribute freely using the encoded diffusion rule. After a single agent finds satisfying executing environment it starts computations.
3. Single solid piece meshes are send to the master application and joined together.

The MG implements evolution and migration cases of the system dynamics, so the main points of 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 from the very beginning - any utilization 'holes' would signal synchronization points or diffusion rule not performing local scheduling properly or being not efficient.

In short, we have formulated the following conclusions concerning the results:

- Diffusion based scheduling allows all agents to spread quickly among computers on the network.The time needed to allocate tasks is small in comparison to the whole computation time.
- Available resources were utilized at 96%, the application used up to 32 agents.
- The load is perfectly balanced according to the given local diffusion law.

It is clear from these conclusions that the diffusion based, local scheduling policy is well suited for such regular problems. The results also confirm that the communication factor may be omitted for certain class of applications, since the total communication overhead is minimal. Also the communication required for the diffusion (examine a VCN node neighborhood and reevaluate the binding energy for every agent) did not degrade the system efficiency.

6.2 HGS

The HGS [4] is a stochastic, hierarchical genetic algorithm optimizing a given function on a defined domain. The application produces agents dynamically in the course of the runtime. A single HGS agent is a container, which starts executing its internal populations as soon as it founds suitable computation environment. The total amount of agents at the execution time may be different even for a particular input data. The algorithm execution path is the following:

1. Create single agent with the initial populations, agent migrates to find suitable execution environment.
2. Agent starts computing and the amount of internal populations changes due to the genetic evolution.
3. If during the computation agent's internal populations amount grows beyond a fixed number, the agent is split and new one is created out of part of the populations set.

The HGS agents perform all three elements of MAS dynamics: evolution, migration and partitioning. The communication overhead (performing agent's migration requires agent's to pass the initial populations) and dynamic scheduling efficiency were tested within this experiment. On a basis of this experiment the following conclusions concerning the diffusion scheduling were drawn:

- Diffusion based scheduling deals properly and effectively with the dynamic rescheduling of the computing units.
- The communication overhead is around 5% of total execution time with application using up to 300 actively computing agents.

Therefore the final conclusion may be stated that the diffusion based local scheduling performs well in case of irregular stochastic problem with dynamic amount of agents. It is very adaptive and local scheduling evaluation does not significantly decrease the solution's effectiveness.

In addition we have also tested the effectiveness of the dynamic, diffusion based scheduling versus centralized, greedy scheduling policy (a Round Robin solution) utilizing low level network message passing mechanisms (Java RMI). Please see [7] for detailed results, which showed that the dynamic scheduling shows moderate speedup loses (up to 30% of total computation time), which disappear when agent's amount grows - in case of 300 agents, diffusion based scheduling performs 10% better that Round Robin.

7 Conclusions and Further Research

The diffusion analogy as well as the MAS technology give way to an effective design of 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. The formal description introduced in sections 4 and 5 provides the discrete equation of evolution and the characterization of admissible controls as well as the cost functional for computing MAS. Such considerations put our problem into the solid framework of the stochastic optimal control theory which provides us with tools such as Bellman-type principles allowing us to study the optimality of control strategies as well as the MAS asymptotics.

References

1. Grochowski M., Schaefer R., Uhruski P.: An Agent-based Approach To a Hard Computing System - Smart Solid. *Proc. of the International Conference on Parallel Computing in Electrical Engineering (PARELEC 2002)*, 22-25 September 2002, Warsaw, Poland. IEEE Computer Society Press 2002, pp. 253-258.
2. Uhruski P., Grochowski M., Schaefer R.: Multi-agent Computing System in a Heterogeneous Network. *Proc. of the International Conference on Parallel Computing in Electrical Engineering (PARELEC 2002)*, 22-25 September 2002, Warsaw, Poland. IEEE Computer Society Press 2002, pp. 233-238.
3. Schaefer R., Flasiński M., Toporkiewicz W.: Optimal Stochastic Scaling of CAE Parallel Computations. *Lecture Notes in Computer Intelligence*, Vol. 1424, Springer 1998, pp.557-564.
4. Wierzba B., Semczuk A., Kołodziej J., Schaefer R.: Hierarchical Genetic Strategy with real number encoding. *Proc. of the 6th Conf. on Evolutionary Algorithms and Global Optimization* Łagów Lubuski 2003, Wydawnictwa Politechniki Warszawskiej 2003, pp. 231-237.
5. Grochowski M., Schaefer R., Uhruski P.: Diffusion Based Scheduling in the Agent-Oriented Computing Systems. *Lecture Notes in Computer Science*, Vol. 3019, Springer 2004, pp. 97-104.
6. Luque E., Ripoll A., Corts A., Margalef T.: A distributed diffusion method for dynamic load balancing on parallel computers. *Proc. of EUROMICRO Workshop on Parallel and Distributed Processing*, San Remo, Italy, January 1995. IEEE CS Press.
7. Momot J., Kosacki K., Grochowski M., Uhruski P., Schaefer R.; Multi-Agent System for Irregular Parallel Genetic Computations. *Lecture Notes in Computer Science*, Vol. 3038, Springer 2004, pp. 623-630.