A Computing Agent Framework

Piotr Uhruski, Robert Schaefer, Marek Grochowski
Stanisław Staszic University of Science and Technology, Department of Computer Science
Al. Mickiewicza 30, 30-059 Kraów, Poland
e-mail: uhruski@ii.uj.edu.pl, schaefer@agh.edu.pl, magja@ii.uj.du.pl

Maciej Smołka
Jagiellonian University, Institute of Computer Science
ul. Nawojki 11, 30-072 Kraków, Poland
e-mail: smolka@ii.uj.edu.pl

Abstract

A software framework that speeds up and simplifies the design and the management of large scale distributed computations is presented. The computational tasks are wrapped into mobile agents supported by a dedicated set of servers. A diffusion-based scheduling makes it possible to decompose and allocate the migration enabled computing agents basing only on local information about the computer network resources. The paper presents some architectural assumptions as well as a dynamical model of the computing system obtained in this way. Two sample tests showing the efficiency of such solution are also attached.

Keywords: agents, parallel computations, diffusion scheduling

1. Introduction

One of uncommon architectures of distributed computations is a collection of autonomous, mobile agents. The efficiency of such a solution comes from the easy way of the application design (each task is wrapped into an agent which is going to satisfy its resource requirements) as well as from fast, decentralized scheduling policies (agent partitioning and migration). An agent scheduling strategy suggested in this paper is based on the molecular diffusion phenomenon (see. [4]). A brief report on architectural principles, scheduling algorithms and a behavioral model of such Multi Agent Systems (MAS) will be followed by the sample tests showing the efficiency of such a solution. The paper summarizes our previous research published in e.g. [6, 3, 7, 2].

2. The software architecture

The framework is composed of two basic components: the set of computing agents and the software servers called virtual computational nodes (*VCN*) associated to the physical computing units (e.g. workstations in the computer network).

VCN's are organized by the logical connection topology that defines immediate neighborhood of each server. The set of servers called *Octopus* delivers basic functionalities for computing agents as creation, running, serialization, deserialization and migration. Octopus platform provides also description of relative performance of all VCN, information about connection speed between VCN through logical paths, and current load of each VCN.

The most typical computing agent is called Smart Solid. It is a pair $A_i = (T_i, S_i)$ where: T_i stands for the computational task, which includes all required for computation data and S_i is an agent shell responsible for the agent's specific 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.

In order to perform agent partitioning and migration, task have to allow pausing and continuation of it's computation. Task can be also partitioned into two subtasks, but task partitioning possibility depends strongly on the computational problem to be solved.

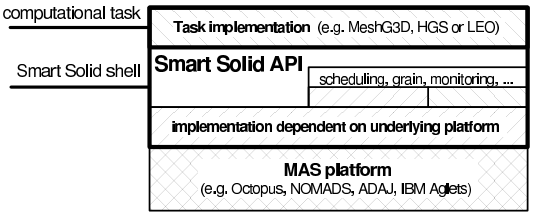

Figure 1. Smart Solid agent architecture.

Smart Solid API (see Figure 1) is a set of interfaces and routines that can be referenced by a computational task in order to access supporting platform and Smart Solid services. Current implementation of Smart Solid API allows to run multi agent application under control of Octopus platform, but chosen architecture enables utilization of other platforms without changes to computational tasks implementation.
Smart Solid shell, by it's implementation, realizes also all agent specific logic such as: searching for resources required by contained task, or balancing the load of computational nodes located in immediate neighborhood of the VCN on which the agent is allocated. These functions are realized by diffusion scheduling using migration and partitioning of application agents.

3. Diffusion scheduling

Diffusion scheduling idea is the most important advantage of our strategy for supporting large scale computations in multi agent systems. Let us introduce the crucial parameters of this strategy:

Binding energy $E_{i,j}$ of the agent A_i allocated on VCN P_j is characterized by the following conditions:
– $E_{i,j}$ is a descending function of E_i and C_i,
– $E_{i,j}$ is a nonascending function of the load on VCN P_j,

Binding energy gradient is a vector:
$\nabla_{i,j} = ((j,l), E_{i,l} - E_{i,j})$:
$E_{i,l} - E_{i,j} = \max_{\zeta \in Q_j}\{E_{i,\zeta} - E_{i,j}\}$, where
Q_j is a set of indices of VCN nodes from immediate neighborhood of P_j that are less loaded then P_j,

Local load concentration of P_j:
$L_j = \frac{E^j_{total}}{perf_j(M^j_{total})}$, where
$E^j_{total} = \sum_{i\in\omega_j} E_i$, $M^j_{total} = \sum_{i\in\omega_j} M_i$,
ω_j – is a set of indices of all agents allocated on VCN P_j,
$perf_j$ – relative performance of VCN P_j strongly dependent on available RAM.

The diffusion scheduling algorithm may be presented in the following way:

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 $\{A_{i_j} = (T_{i_j}, S_{i_j})\}$, $j = 1, 2$;
    disappear
  end if
end if

The above algorithm is processed by the Smart Solid agent shell just after agent creation, after each migration between VCN nodes, and also when underlying platform notifies significant change of local resources utilization. If any of the diffusion algorithm operation can not be performed completely then this algorithm is interrupted and agent is continuing contained task computation. Such condition prevents an agent from going into unexpected state caused by some denial circumstances, like task inability to perform partitioning.

4. The MAS dynamics

The diffusion-based scheduling strategy has proven to be simple and efficient in practice (cf. section 6.) but we also need a theoretical background allowing us to determine if our strategy is optimal or quasi-optimal in any sense. In this section we state a formal mathematic model for multi-agent computations. This model does not take into account agents' communication needs. For details we refer to [6]. Consider for a while that we are given a space of all possible agents and denote it by $\mathbf{A}$. Denote by $\mathbf{N}$ the set of virtual nodes. 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 and linear, i.e. we know the constants $c_1, c_2 \geq 0$ such that in the case of partition $A \to \{A_1, A_2\}$ we have

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

for $i = 1, 2$. Such an assumption seems realistic, in simple cases we may have $c_i = 1$.
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).
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 $u^1_{ij,t}(W_t), u^2_{ij,t}(W_t) \in [0,1]$ 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 $u^1_{ii,t}(W_t), u^2_{ii,t}(W_t)$ 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:

$$\begin{cases} W^i_{t+1} & = F^i_t(\tilde{W}_t) + c_1\ u^1_{ii,t}(W_t)\ W^i_t \\ & \quad + \sum_{j\neq i} u^1_{ji,t}(W_t)\ W^j_t \\ W^{N+i}_{t+1} & = F^{N+i}_t(\tilde{W}_t) + c_2\ u^2_{ii,t}(W_t)\ W^{N+i}_t \\ & \quad + \sum_{j\neq i} u^2_{ji,t}(W_t)\ W^{N+j}_t \end{cases} \tag{1}$$

for $i = 1, \ldots, N$, where

$$\begin{cases} \tilde{W}^i_t & = \left(1 - \sum_{k=1}^N u^1_{ik,t}(W_t)\right) W^i_t \\ \tilde{W}^{N+i}_t & = \left(1 - \sum_{k=1}^N u^2_{ik,t}(W_t)\right) W^{N+i}_t \end{cases}$$

with initial conditions $W_0 = \hat{W}$. It follows that our W_t is a *controlled stochastic process* with a *control strategy*

$$\pi = (u^1_t, u^2_t)_{t\in\mathbb{N}}. \tag{2}$$

5. The optimal scheduling problem

Given the state equations we can formulate *the optimal scheduling problem*. Let the set of admissible control strategies $\mathbf{U}$ be defined as in [6]. Let $V(\pi; \hat{W})$ denote a cost of applying the strategy π when the initial state is $\hat{W}$. We search for a control strategy $\pi^* \in \mathbf{U}$ such that

$$V(\pi^*; \hat{W}) = \min\big\{V(\pi; \hat{W}) :\ \pi \in \mathbf{U},\ W_t \text{ is a solution of (1)}\big\}. \tag{3}$$

Next consider three examples of cost functionals which seem appropriate for multi-agent computations. The first one is the expected total time of computations

$$V_T(\pi;\hat{W}) = E\big[\inf\{t \geq 0 : W_t = 0\} - 1\big]. \tag{4}$$

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

$$V_L(\pi;\hat{W}) = E\left[\sum_{t=0}^{\infty}\sum_{i=1}^{N}(L_t^i - \overline{L}_t)^2\right] \tag{5}$$

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.
The last example allows us to take into account the cost of migrations. Namely take $\varrho_{ij}^m \geq 0$ and put

$$k_M(s,\alpha^1,\alpha^2) = \varphi(s) + \sum_{m=1}^{2}\sum_{i\neq j}\varrho_{ij}^m(\alpha_{ij}^m)^2$$

and

$$V_M(\pi;\hat{W}) = E\left[\sum_{t=0}^{\infty} k_M(W_t, u_t^1(W_t), u_t^2(W_t))\right]. \tag{6}$$

In all above cases under some natural conditions we can show the existence and uniqueness of optimal strategies. Moreover we can also characterize these solutions by means of Bellman-type optimality conditions. The complete considerations can be found in [6].

6. Experiments

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 [1, 5, 3] while [8] contains the Octopus implementation details.
In this paper we present these applications from the system dynamics point of view, namely how the diffusion based scheduling supports different application execution schemas.

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 has 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 Octopus 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.

In short, we have formulated the following two major conclusions concerning the results. First, the 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. And second, the available resources were utilized at 96%, the application used up to 32 agents with the neighboring machines load difference being not greater than 1. That means the load has been perfectly balanced according to the given local diffusion law.

6.2. HGS

The HGS [5] 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 test showed that the diffusion based scheduling deals properly and effectively with an adaptive, dynamic rescheduling of the computing units. The HGS application produced up to 300 actively computing agents with the communication overhead being around 5% of total execution time.
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 therefore 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 [5] 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 than Round Robin.

6.3. Communication Optimized HGS

The HGS is an application with a dynamic execution schema - agents are created dynamically, the amount of agents and their scheduling schema can not be determined. Although the HGS agents do not use an extensive communication, but we have also checked the diffusion based scheduling properties under bad network resources availability conditions (see [3] for details). For such case, the agents binding energy formula has been extended to also take into account communication needs - agents need to communicate with each other and bad network conditions may influence the overall execution time. The communication factor adds to the diffusion the ability to include two behaviors into the local scheduling policy. First,

the agents prefer the migration direction that brings them closer to the agents with whom they need to communicate. Second, the Octopus provides to every agent the description of its immediate neighborhood network links throughput. Using this information agents elect nodes that are closer in terms of network connection bandwidth. Both presented behaviors may intuitively limit the used VCNs amount - agents will prefer nodes that are closer instead of the ones that are free. The tests showed this behavior but also showed how it influenced overall effectiveness.
Two following figures present the dynamic of the HGS computation with and without the communication element of the binding energy. On both figures the thick line shows the amount of active VCNs while the thin one is the total amount of computing agents.

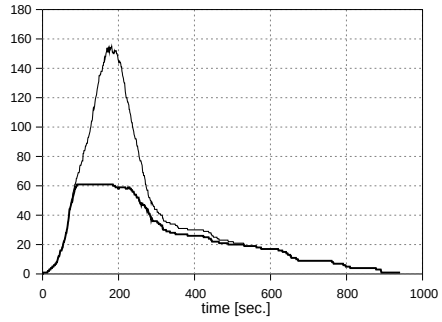

Figure 3. The dynamics of the system utilizing binding energy formula with no communication element (1).

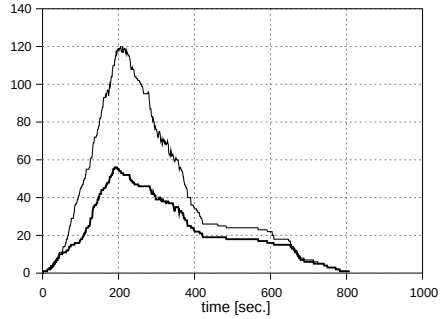

Figure 4. The dynamics of the system utilizing binding energy formula where communication factor is taken into consideration (2).

Please note the amount of used nodes grows slower in the communication enabled experiment as expected. The following table shows the overall computation times in both cases.

Binding energy formula	Agents amount	Total times [sec.]		Parallel computation time [sec.]
		Migration	Communication	
(1)	203	126	29587,2	941
(2)	203	195	26131,2	808

Table 1. The execution times of HGS computations with diffusive scheduling.

As expected, even with smaller amount of VCNs being effectively used, the communication enabled diffusion performs better as diffusion optimizes effectively not only the resources usage, but also the application's communication requirements.

7. Conclusions and further research

The MAS technology speeds up and simplifies the design of large scale distributed computations. Moreover, the diffusion-type scheduling can significantly decrease the execution time. 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 delivers the discrete equation of evolution of such systems, definitions of admissible controls and the cost functional. This approach allows us to characterize the optimal strategies by means of Bellman-type principles.

References

[1] M. Grochowski, R. Schaefer, and P. Uhruski. Diffusion based scheduling in the agent-oriented computing systems. *Lecture Notes in Computer Science*, 3019:97–104, 2004.

[2] M. Grochowski, R. Schaefer, and P. Uhruski. Octopus — computation agents environment. *Inteligencia Artificial*, 2006. accepted.

[3] M. Grochowski, E. Tuska, and P. Uhruski. Influence of inter-agent communication cost to diffusion scheduling in irregular parallel computations. *Inteligencia Artificial*, 2006. accepted.

[4] E. Luque, A. Ripoll, A. Cortés, and T. Margalef. A distributed diffusion method for dynamic load balancing on parallel computers. In *Proceedings of EUROMICRO Workshop on Parallel and Distributed Processing*, pages 43–50, San Remo, Italy, Jan. 1995. IEEE Computer Society Press.

[5] J. Momot, K. Kosacki, M. Grochowski, P. Uhruski, and R. Schaefer. Multi-agent system for irregular parallel genetic computations. *Lecture Notes in Computer Science*, 3038:623–630, 2004.

[6] M. Smołka. Optimal scheduling problem for computing agent systems. *Inteligencia Artificial*, 2006. accepted.

[7] M. Smołka, M. Grochowski, P. Uhruski, and R. Schaefer. The dynamics of computing agent systems. *Lecture Notes in Computer Science*, 3516:727–734, 2005.

[8] P. Uhruski, M. Grochowski, and R. Schaefer. Multi-agent computing system in a heterogeneous network. In *Proceedings of the International Conference on Parallel Computing in Electrical Engineering (PARELEC 2002)*, pages 233–238, Warsaw, Poland, 22–25 Sept. 2002. IEEE Computer Society Press.