Adaptative Partitioning and Dynamic Allocation for Large Computing Systems

Bertil Folliot*, Pierre-Guillaume Raverdy
Laboratoire MASI, CNRS UA 818
Institut Blaise Pascal, Université Pierre et Marie Curie
4 Place Jussieu, 75252 Paris Cedex 05, FRANCE
Tel: +33.1.44.27.34.23 Fax: +33.1.44.27.62.86
E-Mail: {folliot,raverdy}@masi.ibp.fr

Abstract

Dynamic load balancing and migration are two main mechanisms to increase system throughput and to reduce application response time. These mechanisms become very complex (and time consuming) to manage, when the computing system increases in scale and heterogeneity. We have proposed the Territory Model in [14] that separates the system in Computational Domains, where each domains manage its resources by means of Resource Territories, and its application by means of Execution Territories. This paper presents performance evaluations of our model, applied to parallel machines management. The load balancing algorithm associated to each Execution Territory is based on force computation between processes and nodes. Measures are compared with random and diffusion algorithms, and show that our model is efficient, adaptable and scalable with a low system overhead.

1 Introduction

Dynamic process migration is the main load balancing mechanism to react to changes both in computation pattern and in system environment [10]. Migration of non communicating processes, by means of dynamic load balancing, based on computational needs is a well known and difficult problem [4]. Taking into account usage of resources (such as communication between processes) make migration algorithms more intricate. They need to consider at least, the communication patterns of the whole application, as well as the global state of the system when deciding: *when to migrate, which process(es)* and *where*. Classical approaches for resources management in local area networks are not adapted for the following reasons: either a huge amount of state information is collected, and leads to prohibitive algorithms (in term of processors, network or memory consumption) [4], or a limited amount of information is collected, that restricts the supported applications [6]. This is mainly due to the strong interconnection between resources management and application management.

In the Territory Model (TM) [14], one application is associated with one Execution Territory, and resources management is operated by Resource Territories. This model provides a two levels load balancing management. Firstly, inside a Territory, process migration allows to react to temporal changes in computation behavior of the associated application. Secondly, among Execution or Resource Territories, which gains or looses resources according to the application or system evolutions. This scheme reduces the amount of state information collection, the overhead of load balancing management and allows to adapt load balancing decisions inside a Territory to application needs and system behaviors.

*Author's affiliation: UFR d'Informatique, Université Paris VII

In this paper we focus on the performance evaluation of the TM assigned to parallel machines management. This evaluation is based on a parallel simulator implemented on top of the Fujitsu AP1000 distributed memory parallel machine [11]. The simulator is distributed on all the nodes of the parallel machine and takes into account a large number of parameters and evolution laws. It allows to modelize various load balancing algorithms or system configuration, and also the execution of various classes of parallel applications. Each application is defined by its initial number of processes and its evolution laws. Each process is defined by its resources consumption (in term of processor use, memory needs and network traffic) and the evolution laws of these resources consumption. Load balancing management inside an Execution Territory is carried out by *forces* between applications components, as in CFM [15], EOS [9] or Bellorophon [3]. Concurrent accesses to the same resource (e.g. processor) define *repulsion* forces between processes whereas communication defines *attraction* forces. The load balancing mechanism issued from this model is efficient and can operate at various level of parallelism granularities. Performance comparison with two other load balancing managements, Diffusion (designed for parallel machine) [16] and Random, show that the Territory Model is stable and scalable. The proposed algorithm gives good performance results for the studied applications with a low system overhead.

The paper is organized as follows. Section 2 introduces the basic concepts of the Territory Model. Section 3 describes the implementation of the parallel simulator and the various application parameters. Section 4 presents and compares performance results of our model. Finally, we conclude in Section 5.

2 Territory Model

Our Territory Model is composed by three entities: Computational Domains, where cooperate Resource Territories (RT) and Execution Territories (ET). A computational domain gathers hosts of a distributed system, or processors of a parallel machine and it cooperates with others domains. RTs gather identical resources (servers, processors, peripherals, ...) and allocate them to applications. One ET is assigned to the execution of one application. In this section, we describe only the ET part and the RT part (called Idle Territory) assigned for parallel machines management. The complete description of the TM can be found in [13].

When an application starts, its initial ET contains an initial application configuration (or a default set of nodes if no initial configuration is given). To respect *space-sharing* property which provides efficient load balancing onto massively parallel machines [2], territories cannot overlap. So, each processor in an execution territory only executes processes of the same application.

As the execution territory load or the global system load evolves, the ET gains hosts or gives them back from/to other territories. An ET is defined by three configurable parameters: dynamic evolution, bounds and priority. Dynamic evolution means the maximum number of resources that can be requested by an Execution Territory in case of overload (0 means no evolution). Bounds are used to limit the maximal number of resources inside an ET. Priorities are needed to solve conflicts between ETs, according to the user or the application.

In this implementation, every territory contains a centralized manager (either a RT-Manager or a ET-Manager). Each ET-Manager collects state information, takes migration decisions and collaborates with the other managers that may exist.

2.1 Idle Territory

The Idle Territory (IT) is the Resources Territory for idle processors in the case of a massively parallel machine. The IT-Manager is in charge of retrieving and providing idle hosts from/to other ET-Managers (if any). Initially, the system contains only one IT. Each manager (including the IT-Manager) communicates through a virtual ring. When an application is launched a new territory is created according to the initial application configuration, then the new manager is inserted in the virtual ring.

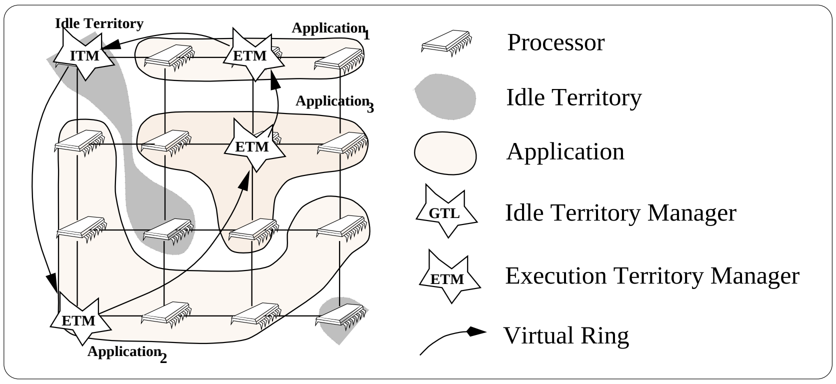

Figure 1: Territory Model

Figure 1 presents an example of the Territory Model for a 4x4 grid parallel machine. There are three Execution Territories composed of three, four and six nodes. The remaining available nodes belong to the Idle Territory. The four managers exchange their average load information via the virtual communication ring in order to get available nodes or to give them back.

The IT-Manager is the only manager that has a global knowledge of the underlying communication network and machine architecture. ET-Managers just had knowledge of their application execution and behavior. Thus, the IT-Manager is in charge of choosing the most efficient idle node to a requesting ET-Manager.

2.2 Statistics monitoring and execution graph

Statistics monitoring is performed on each host by a Local Statistics Server (LSS). It periodically asks the local system for information on the operations performed by the local processes (related to processor, memory and communication). When this information is requested by the ET-Manager, the LSS groups the answers and subsequently sends to the ET-Manager a statistics message. The LSS also sends a message to the ET-Manager in case of a local overload, according to various configurable thresholds (e.g. cpu load, swap, available memory).

Periodically or in response to a signal of an overloaded node, the ET-Manager asks for all the statistics in its territory. Local overload detection classically relies on two thresholds $Tmin$ and $Tmax$ between idle and medium states and between medium and overloaded states [7]. These thresholds are not directly used to decide placement or migration but help to select or discard decisions. Other thresholds are also defined for memory use and communications.

After collecting the statistics, the ET-Manager computes the *execution territory graph* that represents local resources consumption (processor and memory) and remote resource consumption (communications). The execution territory graph is used by the ET-Manager to find out a balanced application configuration inside its territory, by means of *forces* evaluation: concurrent access to the same local resource by two processes tends to separate them (*repulsion*), communicating processes tend to attract themselves on the same node or on closely related nodes (*attraction*), while the amount of memory used by a process tends to stick it on the same processor (*inertia*).

The ET-Manager groups processes, depending on their communication level, and then evaluate if the migration of these groups may improve application execution. Migration may be performed towards hosts already inside the existing territory, or towards new idle hosts that should first be requested to the IT-Manager. In such case, the IT-Manager selects some appropriate hosts according to processes needs and communication delays (distance between idle hosts and the requesting territory). Note that the Idle Territory Manager may limit or refuse requests based on the dynamic evolution and the bound parameters.

When a node inside a territory becomes idle, the ET-Manager may either keep it for further territory needs or give it back to the IT-Manager. When an application ends, all the nodes of the territory are given back to the IT-Manager. In both cases, if the Idle Territory is not empty, the host is inserted into it. Otherwise, the idle host becomes the new IT-Manager.

When the Idle Territory does not exist anymore (i.e. all nodes are used by several territories), an ET-Manager from an overloaded territory requests hosts directly to the other ET-Managers. Each ET-Manager evaluates if it can give a host, based on its priority-load ratio. If no positive answer is received, the overloaded territory has to keep its work until the situation evolves. One possible solution would be to manage a batch queue in order for some low priority applications to wait the end of high priority ones.

2.3 Comparing Territory Model with Diffusion Algorithm

In the Diffusion algorithm [12], each node cooperates with its neighbors nodes. They exchange information and, in case of a local overload, processes allocation and migration are performed on an unloaded node. If no neighborhood is underloaded, the process allocation request is transmitted to the closest underloaded node following a decreasing load path. In the Paros system [12], a local overload is detected when there is at least 2 processes in the processor waiting queue.

The main advantage of this algorithm is that it is fully distributed (no coordinator), and thus extensible to a large number of nodes. It's main drawback is that it does not take into account communications locality between programs. Thus, highly communicating programs may be allocated on very distant node.

3 Parallel Simulator

We have implemented a parallel simulator to evaluate our algorithm on top of the Fujitsu AP1000 distributed memory parallel machine [11]. This machine is composed of 64 cells (Sparc processor) each having 16 Mbytes of main memory. Communication between cells is done by message passing. One back-end node control the machine.

Our simulator is composed by two main programs:

- One program for the back-end node controls the initialization of the parallel machine, starts new parallel simulations, allows to visualize the various territories evolutions and keeps trace of the simulation results.

- One program for each of the 64 nodes executes a Local Statistic Server, and may also execute a Execution Territory Manager or the Idle Territory Manager. Selections of the IT-Manager and ET-Managers are done by the control program.

The simulator may be configured to simulate any kind of parallel machine, or heterogeneous distributed systems in terms of processor speed, amount of memory, network topology and speed.

A parallel application is defined by its initial number of processes, its initial node configuration, and its dynamic evolution laws in terms of process creation and destruction (probability laws according to application execution time). A parallel application process is defined by its lifetime, processor usage, amount of memory and a list of communication patterns with other processes. For these three parameters, dynamic evolution laws may be added (also probability laws according to process execution time). Finally, a given simulation is defined by the number of parallel applications to be executed and the definition of each application.

4 Performance Measurements

The simulator has been configured to simulate a 8x8 Torus distributed memory parallel machine with message passing communication. Processor speed, memory and network performances are homogeneous. However, communication delay between two nodes slightly depends on the distance between them. Link throughput is 10 M-Bytes/s.

We have defined three classes of applications: *CPU intensive, 9/10 CPU-1/10 communication,* and *communication intensive* (half CPU-half communication). For each class of application, three evolution laws exist in term of process creation and destruction: *static* (no evolution), *normal* and *highly dynamic*. An application is also defined by a memory range. Each process uses an amount of memory selected by a random choice into that range.

In Section 4.1 we describe the five applications used for our measurements. We then compare the performance of three major application classes and investigate the influence of the application concurrency, the initial territory size, and the communication bandwidth. Section 4.6 indicates the load balancing overhead of the Territory Model and the force algorithm.

4.1 Applications Description

Table 1 presents the five applications (A..E) used in the followings. Applications A, B, and C are used to evaluate and to compare the performance of the Territory Model with Random and Diffusion. They represent the three classes of evolution laws (static, normal, high). Applications A and B are CPU intensive. Application C is 9/10 CPU-1/10 communication. Application D (normal evolution class, CPU intensive) is used to determine the influence of both the initial execution territory size and the application concurrency (i.e. the total number of processes remains constant but the number of application vary). Application E (highly dynamic, highly communicating) is used to evaluate the influence of the communication bandwidth.

The five applications have the same priority and no bounds are set. Except when indicated otherwise, the initial execution territory size has been set to three nodes, and the dynamic evolution rate to one node.

4.2 Performance of the Territory Model

Results in Table 2 gives the mean execution time (ratio between lifetime and execution time of processes) for ten identical applications started simultaneously on the 8x8 torus parallel machine. Lower is the result (minimum 1.0) better is the performance.

For the B application class, the mean execution time of the Territory Model is slightly under the Diffusion algorithm. Indeed, this is the worst case for the ET-Manager where the locality of process execution is not an important concern due to the very low level of communications.

Table 1: Applications Description

Appli.	Initial # of proc.	Appli. Class	Memory (Mb)	Evolution law
A	6	CPU	1	static
B	12	CPU	1	normal low
C	10	1/10 COMM	0.5-4	high
D	12	CPU	1	normal low
E	12	1/2 COMM	0.3-3	high

Table 2: Mean Execution Time

Application	# appli.	System load	Territory	Diffusion	Random
A	10	low	**1.27**	1.71	1.72
B	10	normal	2.06	**1.96**	2.49
C	10	high	**1.67**	2.14	2.64

Moreover, in this simulation, we have limited the dynamic evolution of a territory to one node (i.e. only one node is granted by the IT-Manager at each request). As all the processes are CPU intensive, we expect better results if the Idle Territory Manager gives a set of idle hosts instead of one.

The A and C application classes shows the very good performances of the Territory Model. For application A there is a total of 10 applications each running 6 processes. This gives 60 processes running on 64 processors. The local overload threshold used in the Diffusion algorithm, do not take into account the global underload of the system. This inadequacy is also shown by the performance of the Random algorithm that is slightly the same than Diffusion. The result of our algorithm for the C application class outperforms the other results. As the territory sizes are dynamically increased according to the application locality, the communication delays are severely limited. As for application B, results should be even better if the dynamic evolution rate is increased. Application C is one of the worst case for the Diffusion algorithm. Indeed, idle nodes allocation follow a lightest load path that may find very distant node thus increasing greatly communication delays.

Figures 1, 2 and 3 present the node occupancy for applications A, B and C. Vertical axe gives the average processor usage in percentage (more than 100% means the host is overloaded). Horizontal axe represents the cumulative number of hosts above the corresponding processor utilization point. Hosts are sorted in descending order of load.

For application B, there are 10 processors which have a load above 200% for the Territory Model, 180% for Diffusion and 225% for Random. We can observe that the maximum overload for the Territory Model (2 hosts at 300%) is superior to the maximum of Diffusion (9 hosts between 200 and 220%). This is mostly due by the Territory Model itself where migration implies first to increase the size of the territory. At the opposite, Diffusion uses only 55 nodes while Territory Model uses all the 64 nodes.

For application A, once again Territory Model tries to use all the nodes of the system, with 20 nodes having less than 25% of load. Diffusion only uses 40 nodes. In that case, the Territory Model gives better results as very few nodes are overloaded (2 nodes).

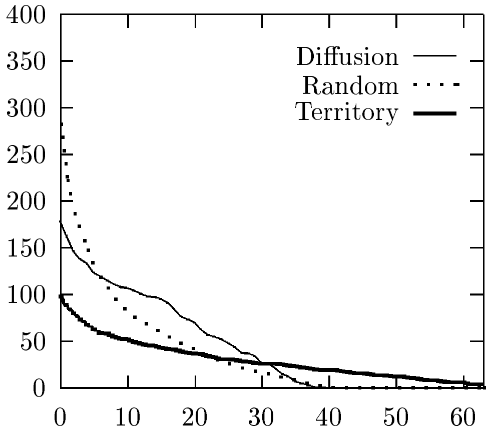

Fig. 1: Application A

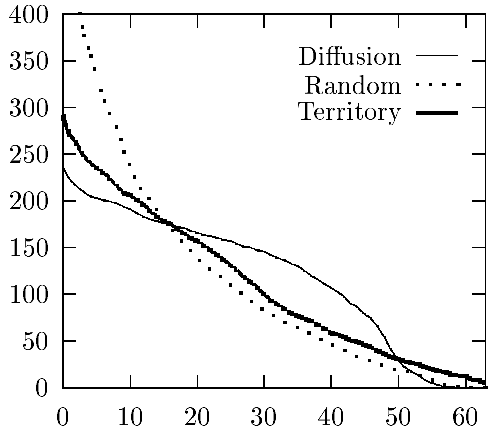

Fig. 2: Application B

Table 3: Mean Execution Time

# Appli.	# Proc.	Territory	Diffusion	Random
5	12	**1.60**	1.76	1.67
10	6	**1.25**	1.72	1.72
20	3	**1.13**	1.76	1.78

Table 4: Migration

# Appli./Proc.	Method	# migration
5/12	Diffusion	30
	Territory	**12**
10/6	Diffusion	13
	Territory	**8**
20/3	Diffusion	9
	Territory	**3**

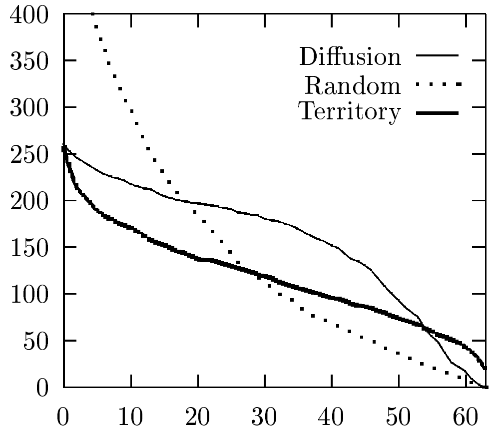

Fig. 3: Application C

For application C, Territory Model and Diffusion use all the available processor, but the load is more evenly distributed for the Territory Model. The node occupancy in figure 3 do not reflect the communication contention in the Diffusion algorithm.

4.3 Influence of Application Concurrency

Table 3 evaluates and compares the influence of the number of concurrent running applications (application class D), given the same global number of processes. There is 60 processes divided in five, ten and twenty applications. The system is globally underloaded (60 processes running on the 64 nodes).

Results for Diffusion and Random are quite the same and do not vary when the number of applications increases. Indeed, these algorithms do not take into account the "application" notion. Performances of the Territory Model increases with the number of applications. In fact, this is due to a side effect: the initial number of nodes for a territory. Remember that we have set this number to three nodes. Thus, for five applications there are 15 pre-allocated nodes, while for twenty applications there are 60 pre-allocated nodes. This pre-allocation is optimal (60 processes running on 60 nodes). The next section evaluates the influence of the initial territory size.

Table 4 indicates the number of migration performed by the Territory Model and the Diffusion algorithms. We observe that due to an initial good locality, the dynamic evolution of application D only incurs a low level of migration in the Territory Model.

Table 5: Mean Execution Time

Appli.	1 node	3 nodes	6 nodes
D	4.58	2.19	**1.68**

The three following Figures (4, 5 and 6) indicate the processor loads for the three experiments with 5, 10 and 20 applications. Once again Territory Model uses all the available 64 processors, while Diffusion uses 34, 43 and 50 processors respectively.

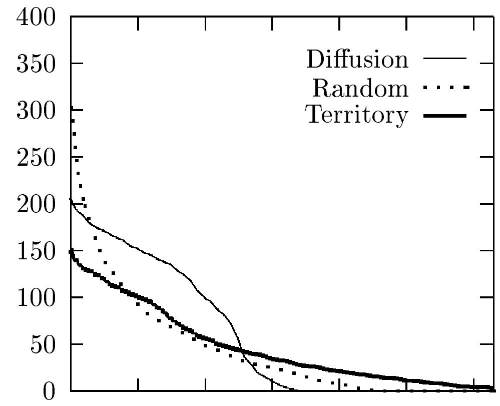

Fig. 4: 5 applications of 12 processes

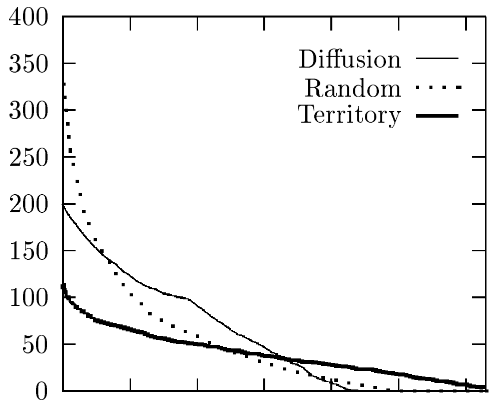

Fig. 5: 10 applications of 6 processes

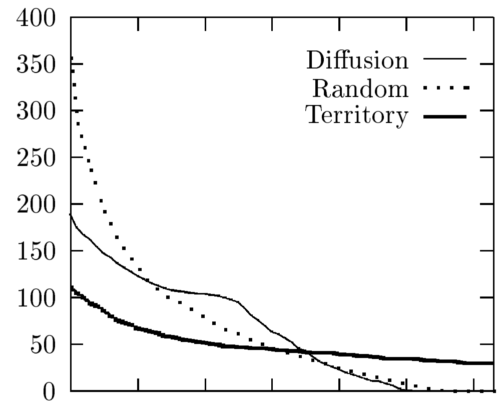

Fig. 6: 20 applications of 3 processes

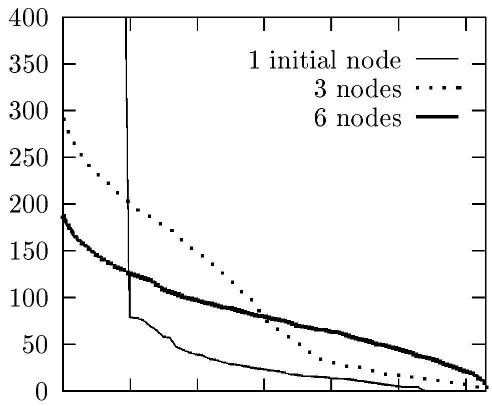

Fig. 7: Influence of the Initial Territory

4.4 Influence of Initial Territory Size

We present the influence of the initial territory size on a given application. Ten applications D are launched in parallel with respectively one, three and six initial nodes.

Table 5 gives the mean execution time for the three initial configuration. As expected, the mean execution time decreases greatly when the initial number of host increase. The processor load is given in Figure 7.

With only one initial node there is ten processors with a huge overload (600 %). These ten nodes corresponds to the initial allocation for the ten territories. With three initial nodes the overload decreases greatly, but the global load is not well distributed. This is due to the time for each territory to get the extra needed nodes. Note that with 6 initial nodes, the initial process allocation is close to the optimal. Indeed, there are 60 nodes initially allocated (6 by applications) on the 64 nodes. Good results presented in Section 4.3 & 4.4 suggests that

Table 6: Mean Execution Time and Average Inter Processes Distance

Appli.	Territory		Diffusion		Random	
	Time	distance	Time	distance	Time	distance
E	**2.22**	0.76	3.46	2.15	2.49	1.43
E'	**2.04**	0.38	3.27	0.68	2.55	2.01

the initial execution territory size is of primary importance and should be adapted by the application itself to its estimate.

4.5 Influence of Communication Bandwidth

This section presents the influence of the communication bandwidth (Figures 8 and 9). Application E (highly dynamic and highly communicating) is described in section 4.1. Communication throughput between two processes vary from 300 KBytes/s to 5 Mbytes/s. To compare performance with a static application, we measured also application E' (static, highly communicating). In that case, communication throughput between two processes is set to 500 Kbytes/s. For this experiment we have the following hypotheses: local communication speed is the fastest, and remote communication speed is proportional to the distance (in term of nodes) between the two processes.

Table 6 confirms the better performance of the Territory Model and the good locality of the communicating processes. The space-sharing being respected, applications execute on distinct partitions of the machine, thus using different links for their communications and avoiding network contention. Note that a random algorithm is better than an algorithm not taking into account the locality of communicating processes.

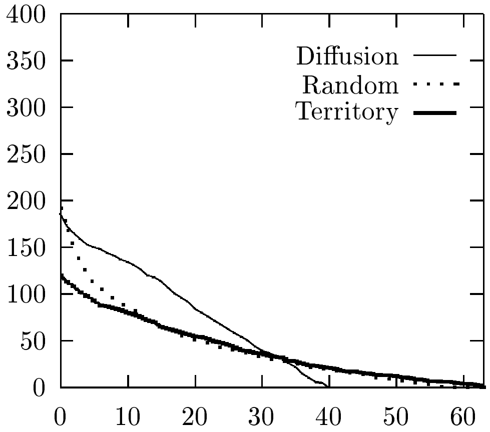

Fig. 8: Application E

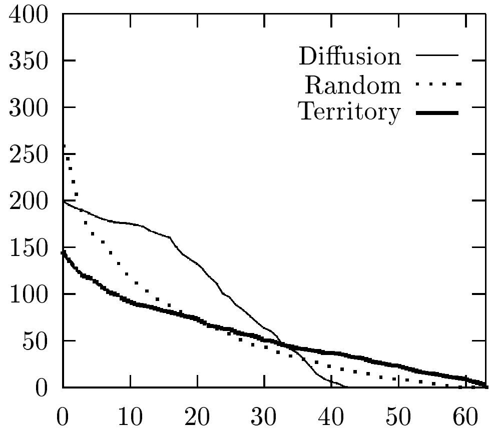

Fig. 9: Application E'

4.6 Load Balancing Overhead

An important potential drawbacks of the Territory Model, compared with Random or Diffusion, are the computational costs of the communicating processes groups and the force algorithms. In this section, we evaluate our load balancing mechanism overhead.

Table 7 gives the time required by the decision algorithm, from processes groups creation, to migration decision. Results do not include statistics retrieval or process migration costs. Evaluations have been performed on a SUN Sparcstation 5.

Each lines gives results for an application composed of nb_{proc} processes, distributed on nb_{node} nodes, and for different number of communication per processes (one, three and five). The

Table 7: Overhead of the Territory Model (second)

# of nodes	# of proc.	# of comm. per process			single process migration
		1	3	5	
5	10	<0.01	<0.01	<0.01	<0.01
	50	0.02	0.02	0.04	<0.01
10	10	<0.01	<0.01	0.01	<0.01
	100	0.03	0.71	0.10	0.05
	500	0.71	1.70	2.70	0.58
50	50	0.15	0.24	0.31	0.14
	100	0.34	0.53	0.64	0.28
	200	0.46	0.83	1.10	0.40
	500	0.70	1.75	2.75	0.62

last result gives the computation time for a single process migration (highly communicating) taking communication pattern into account.

To perform load balancing in less than one second, we can deduce from these results an upper limit of 200 processes per application. Computation time increases very slightly with the number of processes because more nodes in the territory are overloaded. Thus, less nodes are potential receiver for migrating groups, decreasing the complexity of the force computation.

5 Conclusion

In this paper we have argued that the Territory Model is able to manage resources and applications in large and heterogeneous computing systems. We have focused on a parallel machine management by means of TM and presented performance evaluations and comparisons for various application classes. In our model, load balancing operates at two levels: (1) inside a territory to balance the load according to forces between processes and processors, (2) between territories for host sharing. The Territory Model reduces the overhead of the multi-criteria load balancing management (processor, memory and communication), allows a fine monitoring of the system state, and is adaptable to network of workstations, as well as parallel machines. The major problem of diffusion-like algorithms is that they do not try to improve local situation (processor load) if it does not exceed a given threshold. A low threshold means instability whereas high threshold leads to a poor global use of system resources.

To investigate the various classes of applications and their performances in several system configurations, we have implemented a parallel simulator on top of the Fujitsu AP1000 parallel machine [11]. Measurements for a parallel machine of both processor and communication intensive parallel applications show that the territory scheme is stable and scalable. The presented measures particularly shows the good performance of the execution territory for all the studied application classes and particularly for communication intensive applications where node locality is of primary importance.

The close perspectives to our work are twofold. First, an implementation of the Territory Model on a local area network of seven Fujitsu AP1000 parallel machines is under way. Secondly, a load balancing service for network of workstations [8] will be integrated in the Territory Model by means of Computational Domain. Our ultimate goal is to provide a complete platform with a Territory Model based operating system for parallel and distributed load balancing and resource management for high performance applications [5, 1] running on large computing systems.

Acknowledgments

We would like to thank Professor Etsuya Shibayama from Tokyo Institute of Technology and Fujitsu for their help to develop the simulator on the AP1000 machine.

References

[1] Gang Cheng, Kim Mills, and Geoffrey Fox. An interactive vizualisation environment for financial modeling on heterogeneous computing systems. In *Proceedings of the 6th SIAM Conference on Parallel Processing and Scientific Computing*, pages 891–898, March 1993.

[2] M. Crovella, P. Das, C. Dubnicki, T. LeBlanc, and E. Markatos. Multiprogramming on multiprocessors. In *Proc. 3rd IEEE Symposium on Parallel and Distributed Processing*, pages 590–597, Dallas, December 1991.

[3] Peter W. Dickman. Effective Load Balancing in a Distributed Object-support Operating System. In *Proceedings of Second International Workshop on Object Orientation in Operating Systems*, pages 147–153. IEEE, 1992.

[4] Fred Douglis and John Ousterhout. Transparent Process Migration: Design Alternatives and the Sprite Implementation. *Software – Practice and Experience*, 9(4):757–785, August 1991.

[5] Guy Edjali, Nahid Emad, and Serge Petiton. Hybrid methods on network of heterogeneous parallel computers. In *Proceedings of the 14th IMACS World Congress*, Atlanta, USA, July 1994.

[6] X. Evers, J. F. C. M. de Jongh, R. Boontje, D. H. J. Epema, and R. van Dantzig. Condor flocking: load sharing between pools of workstations. Technical Report DUT-TWI-93-104, Delft University of Technology, The Netherlands, 1993.

[7] Bertil Folliot. *Méthodes et outils de partage de charge pour la conception et la mise en oeuvre d'applications dans les systèmes répartis hétérogène.* PhD thesis, Université Paris VI, December 1992.

[8] Bertil Folliot and Pierre Sens. GatoStar: A fault tolerant load sharing facility for parallel applications. In David Powell Eds. Klaus Echtle, Dieter Hammer, editor, *Lecture Notes in Computer Science 852, Dependable Computing - EDCC-1 Klaus Echtle, Dieter Hammer, David Powell Eds.*, pages 581–598, 1994.

[9] Olivier Gruber and Laurent Amsaleg. Object grouping in EOS. In *Proceedings of the International Workshop on Distibuted Object Management*, pages 184–201, Edmonton, Canada, August 1992.

[10] Mor Harchol-Balter and Allen B. Downey. Exploiting process lifetime distributions for dynamic load balancing. Technical Report UCB CSD 95 021, Berkeley, University of California, May 1995.

[11] Hiroaki Ishihata, Takeshi Horie, Satoshi Inano, Toshiyuki Shimizu, Sadayuki Kato, and Morio Ikesaka. Third Generation Message Passing Computer AP1000. In *International Symposium on Supercomputing*, Kyushu, 1991.

[12] T. Muntean and H. Castro. Paros: a parallel operating system for distributed memory parallel computers. In *5iemes Rencontres du Parallelisme*, Brest, Mai 1993.

[13] Pierre-Guillaume Raverdy. *Resource Management and Load Balancing for Large Scale Heterogeneous Systems: Application to mobile and parallel environments (in french)*. PhD thesis, Université Paris VI, June 1996.

[14] Pierre-Guillaume Raverdy and Bertil Folliot. Presentation of the Execution Territory: a Two Level Load Balancing Mechanism. In *European Research Seminar on Advances in Distributed Systems (ERSADS95*, pages 99–104, Alpe d'Huez, Avril 1995.

[15] Mario Tokoro. Computational Field Model: Toward a New Computing Model/Technology for Open Distributed Environment. In *2nd Workshop on Future Trends in Distributed Computing Systems*, Cairo, September 1990.

[16] Chengzhong Xu, Francis Lau, Burkhard Monien, and Reinhard Luling. Nearest-neighbor algorithms for load balancing in parallel computers. *Concurrency: Practice and Experience*, 7(7):707–736, October 1995.