# ENHANCING LOAD DISTRIBUTION STRATEGIES THROUGH SIMULATION

**Alcino Cunha** **Luís Paulo Santos** **Orlando Belo**

Departamento de Informática, Escola de Engenharia, Universidade do Minho, 4710 Braga, Portugal

Email: {alcino,psantos,obelo}@di.uminho.pt

## KEYWORDS

Model Analysis, Artificial Intelligence in Simulation, Discrete Simulation, Intelligent Simulation Environments and Computer Systems.

## ABSTRACT

Load distribution is a well known critical problem in every distributed system. From operating systems to agent oriented applications it is not difficult to find cases where processing nodes are overloaded when, at the same time, other peers present low levels of activity. In agent oriented applications, where the appeal to cooperation is almost a constant event, these unbalanced situations may generate serious cases of contention, deadlock or simply large idle times. The implementation of load distribution strategies in a distributed system may help significantly to improve its overall performance and reduce effectively such undesirable situations. In order to study the effects of different load distribution policies in agent based applications a generic load distribution simulation system was design and implemented. The system allows the specification of multiorganisational distributed systems with dynamic load patterns. Its main characteristics and functionalities are presented in this paper.

## LOAD DISTRIBUTION STRATEGIES ANALYSIS

The quantitative analysis of new load distribution strategies is a fundamental step to assess its initially proposed goals. This analysis must be done prior to the definitive implementation of these strategies, to avoid wasting human and material resources due to an incomplete analysis. Due to its great flexibility, numerical simulation is one of the most used methods to analyse load distribution mechanisms, either to specify the underlying processing system or to specify its workload.

There are two main methods to perform a priori quantitative analysis of load distribution mechanisms: the *analytical* and the *numerical* simulation method. In the former method, where analytical expressions are used to model load distribution mechanisms, there is the possibility of quantifying all fundamental system variables and the knowledge of the expressions that define the relative dependence of those variables (Rodrigues 1988). Most of the models are based on queuing networks to represent the processing system and the executing tasks. Examples of this method can be found in (Eager, Lazowska & Zahorjan 1986)(Theimer & Lantz 1989). The second method, instead of requiring a detailed knowledge of the equations that relate system variables, just requires the knowledge of the internal functioning of the various components of the distributed mechanism. Results are implicitly obtained as a consequence of the interaction between these components, computationally reproduced (Ramamritham, Stankovic & Zhao 1989)(Kremien, Kramer & Magee 1993)(Kremien & Kramer 1992)(Zhou 1988).

However, there are authors that build prototypes of some load distribution mechanisms, to analyse its performance on real physical systems. An artificial, or synthetic, workload is applied to the system, and its behaviour with several different load distribution strategies analysed. These prototypes do not need all characteristics of a real load distribution mechanism, namely those related with liability. Nevertheless, most of the functionality essential to distribution must be implemented (Kunz 1991)(Lüling, Monien & Ramme 1992).

Each of these methods presents advantages and disadvantages:

- due to the difficulty in finding expressions that adequately model the relations between the various system variables, the analytical method is only appropriated for very simple balancing policies and is not well suited for strategies with complex behaviours;
- the analytical method greatest advantage is that results are obtained very fast, even when iterative numerical methods are used;
- simulation with artificial workloads requires a great deal of time to develop the prototypes of the mechanism being analysed, because these must operate on a real system;
- this method implies exclusive dedication of the processing system to the process being analysed. Although this

can be feasible on some parallel machines, it is very difficult on distributed multi-user systems, due to the huge number of users and operating system tasks;

- both the analytical method and the simulation with an artificial workload are very inflexible with respect to the analysis of different physical configurations, due, respectively, to the difficulty of obtaining expressions associated with complex configurations and the difficulty, or impossibility, of changing the physical configuration of the underlying processing system;
- the numerical simulation method requires the development of a simulator that supports the distribution mechanisms being analysed. However, this task is simpler than building prototypes for simulation with artificial workloads; this method is the most flexible with respect to the ability of analysing different system configurations, different load patterns and different configurations of the distributed elements;
- numerical simulation can take a relatively large amount of time if the number of entities involved is large.

Due to the flexibility presented, the numerical simulation method seems to be the best suited to analyse, *a priori*, new load distribution mechanisms. The simulation model hereby presented allows: 1) the definition of the underlying system configuration - being able to simulate parallel or distributed systems on multiorganisational environments; 2) the definition of different configurations of the distribution elements - allowing different arrangements for the various components of the distribution mechanism, namely, the information, transfer, selection and location policies; and 3) the definition of different load patterns. Formally, this model is characterised as (Rodrigues 1988): abstract, numerical, dynamic, explicit and stochastic.

## THE SIMULATION PROCESS

Due to the huge amount of time the simulation process can take to complete, it was divided on two different steps: 1) *the actual simulation*, where results are generated using a given configuration, stored in a file; and 2) *the results analysis*, which is performed later, using the results file generated by the simulator.

With this separation the simulation process can be executed in batch mode. Another advantage is that not all statistics need to be calculated during the simulation, which would be slower and would require the predefinition of all the desired statistics. It would also require a new execution of the simulation each time a different statistic becomes necessary.

All procedures necessary for the simulation process, namely, the simulator itself and the statistical analysis of the results, were developed in SICStus (Carlsson & Widen 1993) Prolog. This language presents some significative advantages, like:

- the ability to model complex algorithmic behaviours;
- significant inference capabilities;
- small development times, which allows a larger number of distribution mechanisms to be modelled, enabling a more effective analysis.

## THE CONFIGURATION FILE

The configuration file is a fundamental element of the simulation process, because it expresses the simulator's flexibility and constitutes the final user interface. The configuration for a parallel machine with 4 nodes and a mesh topology, with load distribution agents using a random strategy. The grammar that rules the configuration file language is presented in figure 1.

```
<configuration> ::= <comment>* <header> <entity>+
<comment> ::=  string
<header> ::= <seed> <begin> <end> <monitoring>
       <control_messages> <message_header>
<seed> ::= seed(integer, integer, integer).
<begin> ::= simulation_begin(integer).
<end> ::= simulation_end(integer).
<monitoring> ::= report_interval(integer).
<control_messages> ::= control_messages_size(integer).
<message_header> ::= message_header_size(integer).
<entity> ::= <comment>|<group>|<node>|
       <communication_line>|<load>|<task>|<agent>
<group> ::= group(identifier, string).
<node> ::= node(identifier,identifier,<identifier_list>,
       <node_resources>,(integer, integer)).
<identifier_list> ::= [identifiers*]
<node_resources> ::= [<node_resource>+]
<node_resource> ::= (identifier, integer, <type>)
<type> ::= active | passive
<communication_line> ::= communication_line(identifier,
       identifier, integer).
<load> ::= load(identifier,<distribution1>,<task_list>).
<distribution1> ::= exactly(integer) | expneg(integer)
<task_list> ::= [<task_load>+]
<task_load> ::= (integer, identifier)
<task> ::= task(identifier,<distribution1>,
       <distribution2>,<task_resources>,(<distribution2>,
       <distribution2>),<distribution1>).
<distribution2> ::= <distribution1> | none
<task_resources> ::= [<task_resource>+]
<task_resource> ::= (identifier, <distribution1>, <type>)
<agent> ::= <aleatory_agent> | ...
<aleatory_agent> ::= aleatory(identifier, identifier,
       integer, integer,<non_empty_identifier_list>).
<non_empty_identifier_list> ::= [identifier+]
```

Figure 1: Configuration language grammar.

To facilitate the simulation process, a syntactic and semantic analysis of the configuration file is performed and the eventual errors presented to the user.

### Simulation control

In this section the parameters of the simulation control predicates common to all entities are presented, namely:

- **Seed.** Adjusts the seed of the random number generator. Essential to perform different simulations with the same configuration, increasing the confidence level on the results.

- **Monitoring start.** Used by the statistical analysis predicates. Indicates from which instant must the tasks arriving to the system be considered, ignoring an initialisation period necessary to achieve a steady state (Rodrigues 1988), essentially with respect to the number of tasks in the system.
- **Simulation end.** Indicates the instant when the simulation finishes.
- **Printing period.** Indicates the time interval between successive printings of the simulation actual time and garbage collection of Prolog's stacks.
- **Control messages size.** Size, in bytes, of control messages sent between the simulation entities, as, for instance, task distribution requests or polling messages. This parameter must be set carefully in order to obtain realistic results.
- **Header.** Size, in bytes, of any message header. Indicates the minimum size of a message circulating in the network. It depends of the communication protocol being simulated.

## Nodes

A node is characterised by the following parameters:

- **Identification.** Unique identifier, that univocally identifies each node in the system.
- **Organisation.** Used to model a system with multiple organisations.
- **Connecting lines.** Defines the network topology and, if the node is connected to various lines, indicates the overheads associated with routing messages destined to other nodes.
- **Resources.** Defines the system's heterogeneity at the processing level. Each resource is characterised by its identification (unique inside each node), its capacity and its type (active or passive). Scheduling priorities are defined by assigning a certain percentage of a resource to a given task. If no assignments are done, each active resource is equitably distributed by the tasks currently being executed. The percentage assigned to each task depends on the total number of tasks. Passive resources can also be shared if its capacity attribute is set to $-1$. The passive resources capacities required by a task are assigned to it during its total execution time.
- **Load pattern.** Used to model the node's load pattern. Each list's element represents a time interval (initial instant, final instant) and the respective load pattern, allowing the definition of a time variable pattern. This is very important to model real distributed systems, characterised by low-level load periods (e.g., at night) followed by overloaded periods (e.g., business hours). The possibility of defining different load patterns for each individual node allows the modelling of, for instance, different peak hours between geographically distant nodes, or even between nodes of the same system.
- **Message processing times.** Indicate the CPU time taken to pack/unpack messages being sent or received by this node and message routing overheads (applicable only if the node is connected to several lines). The indicated values are weights which must be multiplied by the message size; the total time being the time taken by a processor with unitary capacity (to facilitate the comparison between the various nodes times). To allow the definition of heterogeneous systems, these parameters are specific for each node.

## Communication lines

These are characterised by the following parameters:

- **Identification.** Unique identifier, that univocally identifies each communication line.
- **Responsible node.** Defines which node controls the access and rating of the line. Implicitly, it also defines which organisation owns the line.
- **Capacity.** Expresses the line bandwidth in bytes per 1000 simulation units. If each simulation unit corresponds to a millisecond then the capacity is expressed in bytes per second.

The message routing strategy operates as follows. At the beginning each node and line calculates its own routing tables. These tables indicate which line (in case of nodes) or which node (in case of lines) a message must follow to reach another system's node. At the present time these tables are built using the faster path (theoretically) between the calculating entity and all other nodes in the system. This process must be improved in the future, because it doesn't assure the most efficient behaviour in all cases.

The predicate built for sending messages allows multicast, i.e., the set of receivers might be any subset of the universe of nodes in the system. Point-to-point communication results from specifying only one node, broadcast results from specifying all nodes.

## System load

To specify the system load it's necessary to specify the load patterns and the tasks used on those patterns. A load pattern is characterised by the following parameters:

- **Identification.** Unique identifier, that univocally identifies each load pattern.

- **Tasks interarrival time.** Identifies the theoretical distribution that models the average time between tasks arrival to a node.
- **Tasks histogram.** Defines which type of tasks are generated. Each list element indicates the percentage of each type of task.

Tasks types are characterised by the following parameters:

- **Identification.** Unique identifier, that univocally identifies each task type.
- **Duration.** Task duration (theoretical distribution) if it is executed on a node with active resources capacity identical to those presented on a task's resource requirements list. The real execution time calculation on any node is done as follows: first, the smaller ratio between the active resources assigned to the task (be it shared or dedicated) and the respective resources on the task's requirements list is calculated. Afterwards, this ratio is multiplied by the task execution time, to obtain its real execution time. This time is systematically reestimated as tasks enter or leave the node's execution list.
- **Execution deadline.** indicates the maximum time to execute the task, calculated from the instant when the task enters the system. If there is no deadline the parameter `nao_tem` must be used. If there is a deadline, the distribution agent assumes it knows the task's execution parameters.
- **Resources list.** Indicates the resources a node must own to be able to execute the task. The active resources capacity doesn't indicate the capacity that must be allocated to the task. It is only used to calculate the real execution time, as has been explained before. The passive resources must exist in the node and must be allocated to the task during its entire execution time.
- **Communication requirements.** Defines the interval between successive messages addressed to the task's original node and its size. When such a message is sent the task is withdrawn from execution, being scheduled again when an acknowledge message arrives, sent from the original node. This last node only incurs the overheads of packing and unpacking the original message and its answer.
- **Task size.** Size of the task's code and data in bytes. Useful to calculate messages size when tasks migrate between nodes.

## THE SIMULATOR FUNCTIONING

The simulator follows an event-scheduling philosophy (Rodrigues 1988). An entity that generates an event must plan all future events originated by the present one. The main event, which unchains the whole process, is a task arrival to the system. The first arrivals are planned on instant zero of the simulator. Afterwards, each time an arrival occurs, the next ones are planned. This planning method is identical to all other events.

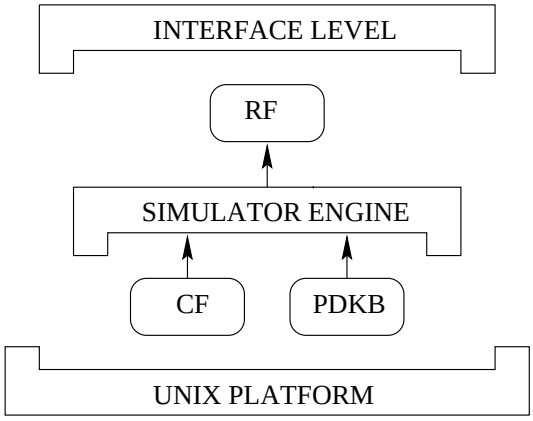


Figure 2: Simulator's architecture overview.

Figure 2 presents an overview of the simulator system architecture, where *CF* stands for *Configuration File*, *RF* for *Results File* and *PDKB* for *Pattern Directed Knowledge Base*.

All active entities (nodes, communication lines and distribution agents) were created using pattern based systems development (Bratko 1990). This approach is ideal to describe reactive entities behaviour, that act according to the detection of events addressed to itself or with interest for its internal processes.

The simulator's kernel was developed using pattern systems (see figure 3). It constitutes the simulation 'engine', managing the events and the progress of time. It is responsible for:

- **Initialisation.** The following actions are performed when a simulation begins: initialise the environment, plan the event that will finish the simulation, plan the first monitoring event and plan the event that will initiate the simulation in all entities. After detecting this event, each entity initialises its own environment and, if it is a node, plans its first task arrival.
- **Monitoring.** Prints the simulation time and performs the garbage collection of the Prolog interpreter stacks.
- **Trigger agents events.** An event is characterised by its description, the instant when it will occur and the list of entities it affects. When the time comes to trigger an event, all entities present in its list must be activated.
- **Remove agents which do not affect entities.** To optimise the simulator, all events that will no longer affect any entity are removed.
- **Time progress.** When, for the actual instant, all events have been processed, the simulation time is advanced to the next planned event instant. Using event-scheduling it is guaranteed that nothing happens meanwhile.

- **End.** When the finishing event is triggered in all entities, the kernel itself finishes, removing all events planned to occur after this instant.

To assure that the real time of the simulation grows at most linearly with respect to the simulated time, it must be assured that the Prolog interpreter base and its control stacks are kept with almost constant size through the simulation. To assure the former the simulation results are saved to a file as soon as they are obtained. As to the later, the backtracking capability of the Prolog interpreter had to be switched off.

```
% /// Starts simulation process
% ///
simulator ::
       [not(tempo(_))]
       --->
       [semente(X, Y, Z), setrand(rand(X, Y, Z)),
       assert(tempo(0)), assert(tarefa(0)),
       assert(tarefa_aux(-1)), todas_entidades(Todos),
       assert(evento(0, Todos, inicia_simulacao)),
       fim_simulacao(Fim),
       assert(evento(Fim, Todos, fim_simulacao)),
       imprime_tempo(Imp),
       assert(evento(Imp, simulador, imprime_tempo)),
       format('Inicio da simulacao.~n', [])].

% /// Ends simulation process
% ///
simulator ::
       [tempo(T),
       evento(T, [], fim_simulacao)]
       --->
       [retractall(evento(_, _, _)),
       retractall(tempo(_)),
       format('Fim da simulacao.~n', []),
       stop].

% /// Triggers events on agents
% ///
simulator ::
       [tempo(T),
       evento(T, [(_, Tipo_agente)|_], _)]
       --->
       [run(Tipo_agente)].

% /// Increments time
% ///
simulator ::
       [tempo(T),
       not(evento(T, _, _))]
       --->
       [findall(X, evento(X, _, _), Tempos),
       min_list(Tempos, NovoT),
       replace(tempo(T), tempo(NovoT))].
```

Figure 3: Excerpt of the simulator's kernel.

## CONCLUSIONS

In this article was presented a simulation platform to study and analyse different load distribution strategies. Additionally, it allows the specification of multiorganisational distributed systems with dynamic load patterns (Cunha & Belo 1997). Besides enabling the comparison of various distribution strategies, it is also a system administration tool, that allows to test the effects of introducing a new distribution strategy. The simulator may help significantly to improve the overall performance and reduce effectively eventual contention or idle situations that may occur on systems with inadequate load distribution policies. It is planned, for future work, the simulator generalization in order to handle generic resource schedulling problems and create a new integrated tool which allows an easier definition of systems configuration and results analyse.

## REFERENCES


Bratko, I. (1990), *PROLOG: Programming for Artificial Intelligence*, International Computer Science Series, second edn, Addison-Wesley.

Carlsson, M. & Widen, J. (1993), *SICStus Prolog User's Manual*, Swedish Institute of Computer Science.

Cunha, A. & Belo, O. (1997), A multi-agent approach for load distribution in multi-enterprise environments, *in* '15th IASTED International Conference on Applied Informatics (AI'97)', Innsbruck, Austria.

Eager, D. L., Lazowska, E. D. & Zahorjan, J. (1986), 'Adaptive load sharing in homogeneous distributed systems', *IEEE Transactions on Software Engineering* **SE-12**(5), 662–675.

Kremien, O. & Kramer, J. (1992), 'Methodical analysis of adaptive load sharing algorithms', *IEEE Transactions on Parallel and Distributed Systems* **3**(6), 747–760.

Kremien, O., Kramer, J. & Magee, J. (1993), Scalable load-sharing for distributed systems, *in* 'HICSS-26'.

Kunz, T. (1991), 'The influence of different workload descriptions on a heuristic load balancing scheme', *IEEE Transactions on Software Engineering* **17**(7), 725–730.

Lüling, R., Monien, B. & Ramme, F. (1992), A study on dynamic load balancing algorithms, Technical Report TR-001-92, Paderborn Center for Parallel Computing.

Ramamritham, K., Stankovic, J. A. & Zhao, W. (1989), 'Distributed scheduling of tasks with deadlines and resource requirements', *IEEE Transactions on Computers* **38**(8), 1110–1123.

Rodrigues, A. G. (1988), *Simulação*, Universidade do Minho.

Theimer, M. M. & Lantz, K. A. (1989), 'Finding idle machines in a workstation-based distributed system', *IEEE Transactions on Software Engineering* **15**(11), 1444–1458.

Zhou, S. (1988), 'A trace-driven simulation study of dynamic load balancing', *IEEE Transactions on Software Engineering* **14**(9), 1327–1341.