2017 International Nuclear Atlantic Conference - INAC 2017
Belo Horizonte, MG, Brazil, October 22-27, 2017
ASSOCIAÇÃO BRASILEIRA DE ENERGIA NUCLEAR - ABEN

Agência Brasileira do ISBN
ISBN 978-85-99141-07-6
9 788599 141076

DYNAMIC LOAD-BALANCING - EXTENDED GRADIENT MECHANISM - GRAPHIC REPRESENTATION

Francisco J. Muniz

Centro de Desenvolvimento da Tecnologia Nuclear - CDTN,
31270-901 - Belo Horizonte - MG, Brazil,
muniz@cdtn.br

ABSTRACT

Load-balancing methods are quite well described in the open literature (hundreds of articles can be found about this subject). In particularly, about the Dynamic Load-balancing mechanism Extended Gradient (EG), several articles of the author are available. Even though, there are some overlap, each one of them is focused on a particular aspect of the mechanism, in a complementary way. In this article, a graphic representation of the Extended Gradient mechanism is done: this representation way had not yet been explored. However, for an in-depth knowledge of the Extended Gradient mechanism, at least, some other articles should to be read. In the CDTN, Clusters are used, mainly, in deterministic methods (CFD) and non-deterministic methods (Monte Carlo).

keywords: Dynamic Load-balancing, Extended Gradient, Cluster.

1. LOAD-BALANCING MODULE

The graphic interaction among the several processes that represent the Extended Gradient mechanism is shown in Fig. 1. The processes are represented by lines, with arrows indicating its execution flow. The marks (two small concentric circles) show the processes' start positions. Devices marked with "idx" represent the semaphores. A dotted line (in a mirrored "S" shape) shows the interaction between processes of neighbour nodes (node "n" and "n+1"). The rectangles of round corners (marked with "Mx") represent communication between processes, utilizing shared memory. The circle marked with "gm" implements the Gradient Model (GM) mechanism. The gm process interact with several others processes (through the operations P, V and V, P on the semaphores id1/id2), in order to provide the dynamic load-balancing facilities. The gm process, periodically, determine the processor status (which is: (1) the local processor, if it is lightly-loaded; (2) if the local processor is heavily-loaded, the status of the processor is the nearest lightly-loaded processor on the Cluster (if there is one)). When necessary, this status information has to be send the neighbour nodes. The periodicity of the process gm is triggered by the "tick" process, which is, also, represented in Fig. 1. Therefore, to determine the local node status, the gm uses the local node load level together with the neighbour nodes status. The neighbour nodes status are received by the "listen" processes. An idle node is selected by calling the "get_node" function - which is, also, identified in the referred figure. The get_node function wakes up the "egm_in" process through the operations V, P on the semaphore device id3. The egm_in process will ask the local gm process for an available node (processor), through the operations P, V and V, P on the semaphores

id1/id2. If the local node is not idle, but the local gm process knows an available node elsewhere on the Cluster, the egm_in process will inquire the "egm_an" process (through a PVM communication channel) on the node pointed as available. Therefore, the egm_an, by its turn, will check with the gm process on that node to confirm if the node is still available. The processes egm_in and egm_an implements the extension of the GM mechanism: the GM plus the extension is known as the Extended Gradient (EG) mechanism [2]. If, either, approaches "1" or "2" (described before) is successfully, an available node is reserved and its identification is returned to the get_node function, through shared memory (M1), and therefore to the user. If no node proves to be available at the time (both, local or remotely), the egm_in process will wait for a node status update operation, and so on. The selected available nodes have, also, to be marked in a file as reserved, to later on be made available by the Administration Module (which is subject of future work - mentioned in Sect. 3.). The Administration Module will work in collaboration with the "free_node" function (see Fig. 1), which receives as input the nodes name to be made available. The free_node function communicates with the "p_release" process through shared memory (M2). Then, the p_release process cooperates with the "p_enable" process (on the reserved node) to make it, again, available.

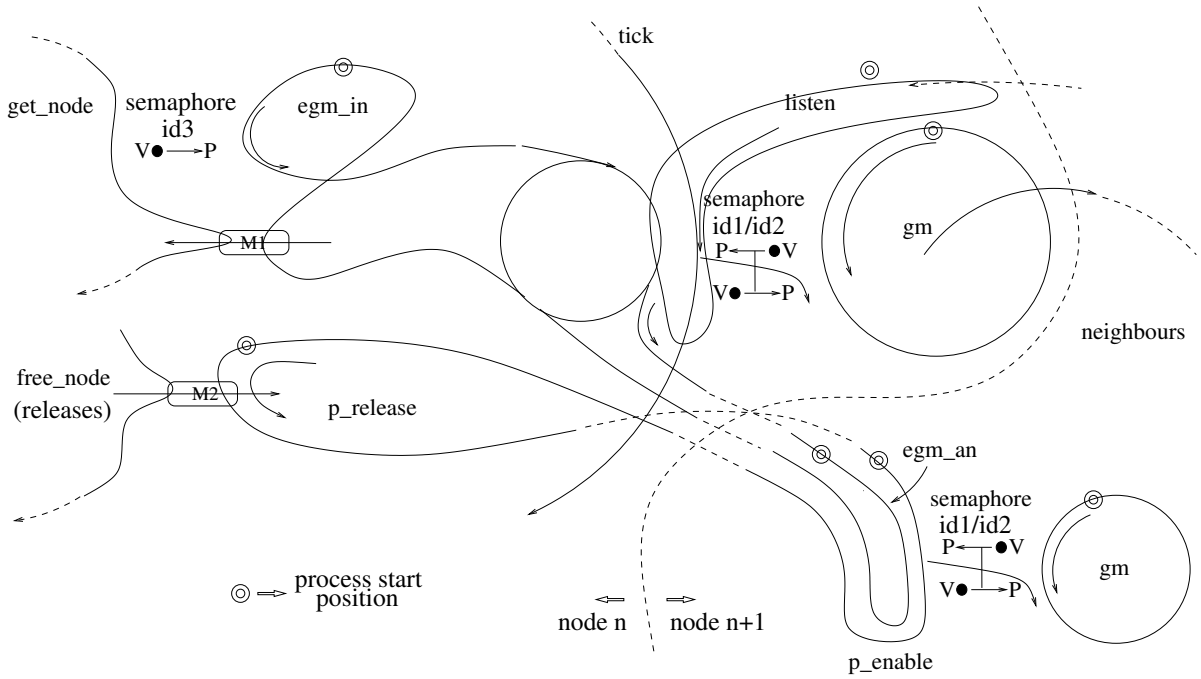

Figure 1. The graphical interaction among the EG processes.

2. BALANCING INTERFACE MODULE

In Fig. 2 it can be seen the interface between the dynamic load-balancing module (the EG mechanism) and the application (the Designer). A set of available nodes can be selected by calling the "hunter" function (hunter(int ∗n_nodes, char node[∗n_nodes][...])). It is needed to inform the number of nodes ("∗n_nodes") to be selected, and a matrix of strings (node[∗n_nodes][...]) to receive the selected node names. As can be seen in the referred figure, hunter process will randomly spawn, in the Cluster, a "h_node" process to each node to be selected. Each h_node process, by its time, will call the get_node function

which will select an available node. In other words, everything that the applications Designer has to do, is to declare the prototype hunter function in its application, call it and link its object code (hunter.o) when compiling the application.

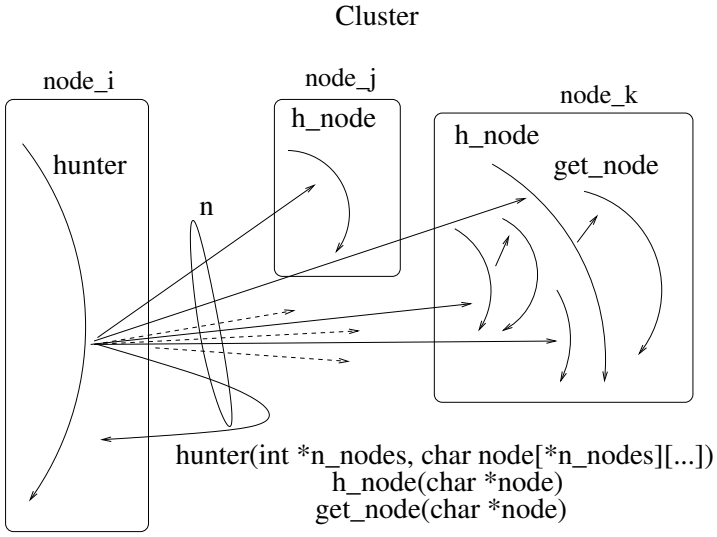

Figure 2. The EG mechanism user interface graphical representation.

3. DISCUSSIONS AND FUTURE WORKS

A dynamic load-balancing system require others two modules, in addition to the module being described in this article, as following. (1) An Administration Module which will determine the submission requirements of a job, from a file in previously agreed format. Requirements, such as: the number of nodes requested; the code to be executed; the input and output files; the queue to which the job (code) is to be submitted (the estimated code execution time); ... Once those requirements were determined, a set of rules (of nature technological and scientific, and of the Cluster's utilization policy) have to be faced with such requirements of submission. If applicable, the needed number of nodes have to be selected by the dynamic load-balancing mechanism. Therefore, the application is mapped to the set of nodes selected. This Administration Module has to be in charge of events generation to the users (i.e. generation of e-mails) in order to keep them informed. Later on, this module has to make the reserved nodes available (when the application which make them reserved ends up) - a module to be developed. (2) And an Occupation Cluster Processing Monitor [9]: such as, a stack of horizontal bars representing the processing intensity of a node set (a bar for each node being monitored). Should, also, be allowed to select the set of nodes to be monitored.

4. CONCLUSION

We hope that this graphic representation will help for a best understanding of the dynamic load-balancing mechanism.

ACKNOWLEDGEMENTS

Financial support for the paper presentation were provided by both: the FAPEMIG ('Fundação de Amparo à Pesquisa do Estado de Minas Gerais') and the CNPq ('Conselho Nacional de Desenvolvimento Científico e Tecnológico') Foundations. I also would like to thank the colleague Vagner Oliveira for his suggestions and comments during the development of this article.

REFERENCES

1. Muniz, F.J.: *Parallel load-balancing on message passing architectures.* Ph.D. thesis, University of Southampton (1994)
2. Muniz, F.J., Zaluska, E.J.: Parallel load-balancing: An extension to the gradient model. *Parallel Computing* **21**, 287–301 (Feb. 1995)
3. Muniz, F.J.: Dynamic load-balancing algorithm porting on MIMD machines. *LCI's 5^{th} Int. Conf.* Austin - USA, May 18-20 (2004)
4. Muniz, F.J., de Carvalho, O.S.F.: Dynamic threshold (over dynamic load-balancing) on MIMD architecture. *Int. Nuclear Atlantic Conf. - INAC.* Santos, SP, Brazil, Sep. 30 to Oct. 5 (2007), ISBN: 978-85-99141-02-1
5. Muniz, F.J.: PVM library spawn function extension. *Int. Nuclear Atlantic Conf. - INAC.* Santos, SP, Brazil, Sep. 30 to Oct. 5 (2007), ISBN: 978-85-99141-02-1
6. Muniz, F.J., de Azevedo, C.V.G., Dalle, H.M.: Node reserve mechanism (over dynamic load-balancing). *Int. Conf. on Mathematics and Computational Methods Applied to Nuclear Science and Engineering - M&C 2011.* Rio de Janeiro, RJ, Brazil, May 8-11 (2011), ISBN: 978-85-63688-00-2
7. Muniz, F.J., de Azevedo, C.V.G., Dalle, H.M.: Dynamic load-balancing system applied to nuclear engineering. *Int. Nuclear Atlantic Conf. - INAC.* Belo Horizonte, MG, Brazil, Oct. 24-28 (2011), ISBN: 978-85-99141-04-5
8. Muniz, F.J., de Azevedo, C.V.G., Dalle, H.M.: Dynamic load-balancing implementation on MIMD machine (and results). *Int. Nuclear Atlantic Conf. - INAC.* Recife, PE, Brazil, Nov. 24-29 (2013), ISBN: 978-85-99141-05-2
9. Muniz, F.J.: Cluster processing business level monitor. *Int. Nuclear Atlantic Conf. - INAC.* Belo Horizonte, MG, Brazil, Oct. 22-27 (2017), ISBN: 978-85-99141-07-6 (submitted).