Classifying Linux Memory Consumption Patterns Based on Self-Organizing Maps

Mauricio T. N. G. Lin[2], Edjard de S. Mota[1], Ilias Biris[2]

[1]Departamento de Ciência da Computação
Universidade Federal do Amazonas (UFAM)
Manaus – AM – Brazil

[2]Instituto Nokia de Tecnologia (INdT)
Manaus – AM – Brazil

{mauricio.lin,ilias.biris}@indt.org.br, edjard@dcc.ufam.edu.br

Abstract. *The problem of memory exhaustion with restricted memory still sparks long discussions inside the Linux kernel developers community. The current implementation to treat the memory exhaustion in Linux is named as Out of Memory Killer and its algorithm to select processes for termination requires more investigation in terms of memory consumption behaviour. This paper describes a methodology to classify memory consumption patterns of Linux applications based on a neural network model known as Self-Organizing Maps. A tool was developed to provide the practical opportunity to apply this mechanism for the classification of memory consumption patterns related to real Linux applications use cases.*

1. Introduction

The Linux operating system is increasingly supporting features of embedded computing [Singh 2004]. Although new improvements are developed for embedded systems, the problem of memory exhaustion in Linux and the current approach to solve it, aptly named as *Out of Memory (OOM) Killer*, has sparked long discussions at Linux kernel community. OOM is even more critical on embedded systems which have little main memory and no swap space, such as palmtops and cell phones [Lin et al. 2005].

The process selection algorithm of OOM Killer, which selects which processes are going to be terminated when OOM conditions occur, could be further investigated in terms of classes of memory consumption patterns. The lack of scientific works related to memory exhaustion in Linux motivated us to develop a methodology to identify and classify the behaviour of memory consumption in Linux. In this paper we summarize such methodology on desktop computers and the experimental results described in [Lin 2006].

The analysis of our method reveals an open road of experiments on embedded systems. The proposed methodology is based on *Self-Organizing Maps (SOM)*. SOM can provide a way to classify memory consumption patterns thus helping the verification of which applications are responsible for memory allocations that may eventually bring a Linux-based system to OOM condition. The use of SOM provides important results since it can represent large quantities of memory consumption collected data on a topographic map. Using a topographic map permits us to trace the progress of memory consumption

for an application, thus leading to an identification of its memory consumption behaviour related to an use case scenario.

The rest of this work is structured as follow. In Section 2 we briefly introduce the concepts of SOM. In Section 3 we give a short description of our method. In Section 4 we summarize the experimental results. Finally, in Section 5 we address our conclusions.

2. Self-Organizing Maps - SOM

SOM is an unsupervised artificial neural network that consists in a lattice of artificial neurons whose weights are adjusted to match input vectors provided in a training set [Borgelt 2000, Honkela 1997].

SOM neurons are placed on a lattice usually 2-dimensional. Neurons become selectively tuned to various input patterns or stimulations during the learning or training process. During tuning the neurons are placed in an ordered manner on the lattice creating a meaningful coordinate system for different input features. When the training is complete, a topographic map of input patterns is formed [Borgelt 2000, Haykin 1999]. The initial synaptic weights of artificial neurons are set arbitrarily. Once all neurons have their weights initialized properly, three important operations occur during the formation of SOM [Haykin 1999]. These are described below:

- *Competition:* For each input pattern, the artificial neurons compute their corresponding values based on a discriminant function. The return value of this function provides the means for the competition between neurons. The neuron with highest discriminant function value is the winner.
- *Cooperation:* The winning neuron establishes a spatial location of topological neighborhood of active neurons, providing in this manner the cooperation among neighbouring neurons.
- *Synaptic adaptation:* Excited neurons have their individual values of discrimimant function increased in relation to the input patterns. This is accomplished by adjusting their corresponding synaptic weights.

The SOM technique described here is able to classify memory consumption patterns based on application use cases, since the set of collected data of memory consumption can be used as SOM input data.

3. A Methodology to Classify Memory Consumption Patterns

In terms of amount of memory size allocated and released we cannot have an absolute definition. What is small or large memory are all qualitative measures which depend on the hardware being used and the kinds of applications targeted to run on it. However, independent from the hardware constraints we can have an abstract model of allocated and released memory. This can be classified via the following memory allocation/release patterns: small chunks of memory cells, big chunks of memory cells, alternated chunks of small and big cells, alternated sequences of small chunks followed by a big chunk of cells, alternated sequences of big chunks followed by a small chunk of cells and random senquence of small and big chunks of cells.

Moreover the frequency of memory usage is also included to classify memory consumption patterns, since its indicates how fast the memory chunks are allocated and

released. Hence the relevant variables for classifying memory consumption patterns can be defined as follows:

- Size of physical memory usage;
- *Memory usage variation (MUV)* indicates the frequency of memory consumption. MUV is given by $MUV = \frac{mem_2 - mem_1}{t_2 - t_1}$, where mem_1 and mem_2 are the size of physical memory usage at instant t_1 and t_2, respectively.
- *Rate of memory usage variation (RMUV)* indicates the frequency of MUV. RMUV is given by $RMUV = \frac{muv_2 - muv_1}{t_2 - t_1}$, where muv_1 and muv_2 are the memory usage variation at instant t_1 and t_2, respectively.

Classes of memory consumption patterns are based on size of physical memory usage, MUV and RMUV properties. Each property can be assigned as *Low (L)*, *Medium (M)* and *High (H)* state and represents respectively, the set of small, medium and large numbers in an interval of known values. The size of physical memory usage can, qualitatively speaking, be in any of the states $\{Low, Medium, High\}$. Analogously, MUV and RMUV follow the same qualitative classification and they are divided equally in 6 parts, since their values also include negative numbers.

The data employed to model the memory consumption patterns are represented by the triple $\langle$*memory, muv, rmuv*$\rangle$. Classes that consume high memory have the first element of triple $\langle$memory, muv, rmuv$\rangle$ at state H and is considered *critical* as illustrated in Table 1.

Critical Classes		
$\langle H_{mem}, \pm L_{muv}, \pm L_{rmuv} \rangle$	$\langle H_{mem}, \pm L_{muv}, \pm M_{rmuv} \rangle$	$\langle H_{mem}, \pm L_{muv}, \pm H_{rmuv} \rangle$
$\langle H_{mem}, \pm M_{muv}, \pm L_{rmuv} \rangle$	$\langle H_{mem}, \pm M_{muv}, \pm M_{rmuv} \rangle$	$\langle H_{mem}, \pm M_{muv}, \pm H_{rmuv} \rangle$
$\langle H_{mem}, \pm H_{muv}, \pm L_{rmuv} \rangle$	$\langle H_{mem}, \pm H_{muv}, \pm M_{rmuv} \rangle$	$\langle H_{mem}, \pm H_{muv}, \pm H_{rmuv} \rangle$

Table 1. Triples belong to critical classes of memory consumption.

Classes regarded as potential candidates to reach the high memory comsumption correspond to memory usage at state M, MUV and/or RMUV at state H. If MUV and RMUV are high and the physical memory usage is about to leave from state M and get in the state H, then there is big chance to reach a high state of memory consumption. Classes that hold this behaviour are named as *potentially critical*.

SOM organizes topologically the classes of memory consumption in different regions. Each region corresponds a pattern of memory consumption. Our SOM applied for classification of memory consumption patterns is comprised of critical, potentially critical and stable regions. Stable regions are triples not defined in critical or potentially critical classes. The practical considerations related to this theoretical approach of applying SOM technique for classifying memory consumption patterns are described in Section 4.

4. Experimentation of SOM to Classify Memory Consumption Patterns

Use cases scenario are designed to provide a way to collect data for SOM training and verify the memory consumption pattern for each scenario after the SOM training. Memory consumption patterns are not related just to the type of application, but also the manner it is used in different scenarios. The selected applications for our use cases are: Gpdf PDF

viewer, Galeon web browser, Totem video player, Gedit text editor and Gthumb image viewer.

A graphical interface based on GTK was developed to view the topographic map of SOM before and after the training. Colors are used to exhibit the synaptic adaptation of all neurons in the lattice before and after the SOM training. The colors representation is conceivable because the vector (memory, muv, rmuv) can be mapped to the 3 basic color components (red, green, blue), known as RGB. Each component of the vector (memory, muv, rmuv) is firstly mapped in the interval [0, 1]. Such mapping is mandatory, since the components of neurons weight vector are values in [0, 1] during the SOM training. After the training process, the weight vector is finally transformed in the color vector, composed by RGB elements, and presented to the user visualization.

A graphical tool named as SOM was developed to identify the memory behaviour of application use cases, and we carried out experiments of 13 different use cases. In order to demonstrate how our methodology is able to classify memory consumption patterns, for lack of space, just 2 of them are exemplified in this work as follows. Figures 1 and 2 illustrate how the SOM tool displays gradually the memory consumption progress of Galeon and Gedit use cases, respectively. Galeon use case consists in loading several web pages sequentially. Gedit use case consists in creating a new text file, typing thousands characters and saving the changes to the disk.

The trained SOM for classifying memory consumption patterns is depicted in Figs. 1(a) and 2(a). The critical region is represented by the pink region and includes the instance $\langle H_{mem}, \pm L_{muv}, \pm L_{rmuv} \rangle$, while the stable region are composed by the blue, green, black and magenta regions. Such regions correspond respectively to the instances $\langle L_{mem}, \pm L_{muv}, \pm L_{rmuv} \rangle$, $\langle L_{mem}, +L_{muv}, +H_{rmuv} \rangle$, $\langle L_{mem}, -H_{muv}, -H_{rmuv} \rangle$ and $\langle M_{mem}, \pm L_{muv}, \pm L_{rmuv} \rangle$. The white regions in other figures correspond the area visited by the application use cases that represent their memory consumption profile.

(a) (b) (c) (d) (e)

Figure 1. Memory consumption progress of Galeon use case.

(a) (b) (c) (d) (e)

Figure 2. Memory consumption progress of Gedit use case.

As you can notice in Figures 1(b), 1(c), 1(d) and 1(e), the Galeon use case presents an stable memory consumption, since the visited areas cover just the blue region along its execution time. Gedit use case behaves differently, since the visited areas occupy firstly the blue region, as depicted in Figures 2(b) and 2(c), and gradually the magenta and part of pink regions are occupied later, as depicted in Figures 2(d) and 2(e). So Gedit use case presents an unstable memory consumption as the critical region is also occupied. Hence SOM tool can provide an alternative mechanism to empirically identify and classify the memory consumption patterns of an application use case.

5. Conclusions

This paper presented an approach to classify the memory consumptions patterns by employing a self-organizing neural network. Our SOM tool can be used to analyze the memory consumption behaviour of application use cases and show the scenarios an application that consumes memory excessively. The quality of classification provided by SOM tool depends if the set of data used during the training can cover real use cases. In our experiments the training took around 7 hours for 13 use cases that correspond about 16.000 samples of input data.

We delineate our experiments in running on a Intel Xeon 2.06Ghz machine with 1 GB of RAM memory and 2 GB of swap space, so further investigations on embedded devices are important for extending this work. Our approach does not guarantee a workable OOM Killer process selection algorithm based on SOM, because no experiments were involved in the OOM Killer context. Additional measurements to compare the current selection algorithm of OOM Killer with our methodology are relevant in order to determine which approach is more viable when OOM happens. Thus the integration of our SOM mechanism with embedded Linux OOM Killer is also important, since our motivation for implementing a memory consumption classifier originates from the framework proposed in [Lin et al. 2005].

References

Borgelt, C. (2000). Self-organizing map training visualization. http://fuzzy.cs.uni-magdeburg.de/~borgelt/doc/somd/. School of Computer Science - Otto-von-Guericke-University of Magdeburg.

Haykin, S. (1999). *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 2nd edition.

Honkela, T. (1997). *Self-Organizing Maps in Natural Language Processing*. PhD thesis, Helsinki University of Technology - Neural Networks Research Centre, P.O. Box 2200 FIN-02015 HUT, FINLAND. http://www.mlab.uiah.fi/~timo/som/thesis-som.html.

Lin, M. (2006). Metodologia para classificação de padrões de consumo de memória no linux baseada em mapas auto-organizáveis. Dissertação de Mestrado, Programa de Pós-Graduação em Informática, Universidade Federal do Amazonas. http://www.dcc.ufam.edu.br/~edjard/pesquisa/MSc/Lin/texto/thesis.pdf.

Lin, M., Medeiros, V., Novellino, R., Biris, I., and Mota, E. (2005). Memory management approach for swapless embedded systems. *Linux Journal*, pages 36–43.

Singh, D. I. M. (2004). Embedded linux: The 2.6 kernel is ideal for specialized devices of all sizes. http://www.linux-mag.com/2004-08/embedded_01.html.