

A Self-Organizing Neural Network for Job Scheduling in Distributed Systems

Harvey B. Newman, Iosif C. Legrand

*Charles C. Lauritsen Laboratory of High Energy Physics
California Institute of Technology, Pasadena, CA 91125, USA*

Abstract. The aim of this work is to describe a possible approach for the optimization of the job scheduling in large distributed systems, based on a self-organizing Neural Network. This dynamic scheduling system should be seen as adaptive middle layer software, aware of current available resources and making the scheduling decisions using the “past experience”. It aims to optimize job specific parameters as well as the resource utilization. The scheduling system is able to dynamically learn and cluster information in a large dimensional parameter space and at the same time to explore new regions in the parameters space. This self-organizing scheduling system may offer a possible solution to provide an effective use of resources for the off-line data processing jobs for future HEP experiments.

INTRODUCTION

Finding and optimizing efficient job scheduling policies in large distributed systems, which evolve dynamically, is a challenging task. It requires the analysis of a large number of parameters describing the jobs and the time dependent state of the system. This study uses the MONARC Simulation system [1] to develop an approach to the job-scheduling task in distributed architectures, based on self-organizing neural networks [2]. The use of these networks enables the scheduling system to dynamically learn and cluster information in a high-dimensional parameter space. We have applied this approach to the problem of distributing offline data processing jobs among the worldwide-distributed regional centers in the LHC computing model. These jobs need random access to very large amounts of data, which are assumed to be organized and managed by distributed federations of OODB systems. Such a scheduling system may also help manage the way data are distributed among regional centers as a function of time, making it capable of providing useful information for the establishment and execution of data replication policies.

THE JOB SCHEDULING

A large number of parameters, most of them time dependent, must be used for the job scheduling in large distributed systems. The problem is even more difficult when not all of these parameters are correctly identified, or when the knowledge about the state of distributed system is incomplete or/and known with a certain delay in the past.

A “classical” model

A “classical” scheme to perform job scheduling is based on a set of rules using (part of) the parameters and a list of empirical constraints based on experience. It can be implemented as a long set of hard coded comparisons to achieve a scheduling decision for each job. In general, it can be represented as a function, which may depend on large numbers of parameters describing the state of the systems and the jobs. After a job is executed based on this decision, a performance evaluation can be done to quantify it.

A self organizing model

This approach is based on using the “past experience” from jobs that have been executed to create a dynamic decision making scheme. A competitive learning algorithm is used to “cluster” correlated information in the multi-dimensional input space defined by the parameters describing the systems, the jobs, the decisions and the results.

In the area of competitive learning quite a large number of modes exist which may have similar goals but differ considerably in the way they work or the implementation is done to solve certain problems. In our case, a feature mapping architecture able to identify correlation and cluster data distributed in a high-dimensional input space is done using, a growing self organizing network [2]. The incremental network models do not have a predefined structure (as in the case of self-organizing maps) and the addition and removal of neurons from the structure are done as part of the learning procedure. Known also as a neural gas, due to the fact that it does not have a static topological structure, such structure can offer an effective approach for feature extraction in multi dimensional space. The aim of the learning process is to cluster the input data into a set of partitions such that the intra-cluster variance remains small compared with the inter-cluster variance and to estimate the probability density function. This clustering scheme seems possible as we expect a strong correlation between the parameters involved. The strategies in modifying the network topology during the learning process (adding, deleting and clustering neurons) are to improve the pattern identification in the data and at the same time to try to increase the entire entropy. The processes of clustering neurons during the learning may be compared with creating molecules in a gas.

A very simple toy example

This over-simplified example aims to schematically present the way a self-organizing network can identify correlations and may be used to cluster the information. We assume that the time to execute a job in the local farm having a certain load (α) is:

$$T_1 = T_0(1 + f(\alpha)) \quad (1)$$

Where T_0 is the theoretical time to perform the job and $f(\alpha)$ describes the effect of the farm load in the job execution time. If the job is executed on a remote site,

an extra factor ($\beta > 1$) is introduced increasing the turnaround time:

$$T_r = T_0\beta(1 + f(\alpha_r)) \quad (2)$$

The ratio between the assumed execution time and the theoretical one (T_0) for a logarithmic load function is presented in Fig. 1.

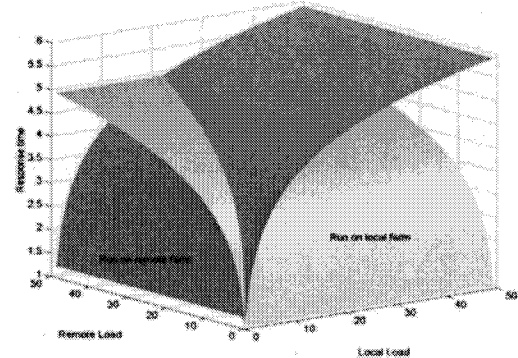


Figure 1. A simple example of the execution time for jobs executed on local or remote systems having different load factors.

The way a competitive learning algorithm for a neural gas like system performs for this problem is shown in Fig. 2. “Neurons” connected by lines are relatively close in the parameter space.

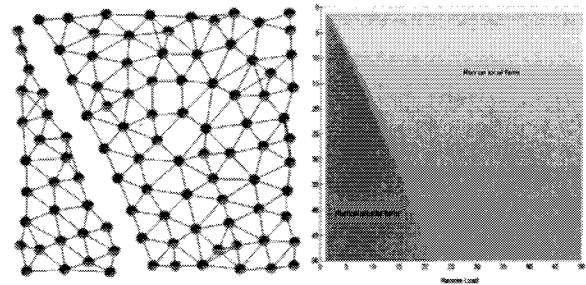


Figure 2. The self-organizing clusters compared with the projected decision types to obtain the best response time.

The Scheduling Decision

The decision for future jobs should be based on identifying the clusters in the total parameter space, which are close to the hyper plane defined in this space by the subset of parameters describing the job and the state of the system (parameters known before the job is submitted). In this way the decision can be done evaluating the typical performances of this list of close clusters and choose a decision set which meets

the expected / available performance / resources and cost.

However, the self-organizing network has, in the beginning, only a limited “knowledge” in the parameter space and exploring efficiently other regions is a quite difficult task.

In this approach for scheduling, the difficult part is not the learning from previous experience, but making decisions when the system does not know (never tried) all possible options for a certain state of the system [3]. Even more difficult is to bring the system into a certain load state, which may require a long sequence of successive decisions taken to achieve it (like in a strategic game problem). The result of each decision is seen only after the job is finished, which also adds to the complexity of quantifying the effect of each decision. For this reason a relatively short history of the previous decisions taken by the system is also used in the learning process. We assume a relatively long sequence of decision (a policy) can be described with a set of a few points decision history, which partially overlap. This means that we can build a trajectory in the decision space by small segments that partially overlap.

An Example

We consider three Regional Centers (“caltech”, “kek”, “cern”) connected. For two of them (“caltech” and “kek”) during a certain period of time, every day, the number of data processing jobs submitted for execution exceed the locally available processing power while the third one (“cern”) does not have any activity. At Day=0 (initial condition) a “classical” scheduling algorithm is used and all the jobs are submitted only to the local regional center. As the rate of sending jobs into the local regional center during a certain period of time exceed the local available processing power, many jobs are put into an execution queue. This job queuing makes the mean value for the turnaround time to be quite modest. In the case a job is exported to another regional center, we considered that the job works with data from its original Regional Center. Due to bandwidth limitation and much higher round trip time (RTT) the execution time on a remote site is longer than running it locally when the processing power is not overloaded. Therefore there is a penalty when jobs are executed on a remote site.

The decision making scheme together with the self organizing networks are “learning” what happens if jobs are exported, and try to provide a better mean return time for all jobs in each Regional Center After

several days in which the usage pattern was similar as on the first day, the self-organizing scheduler has investigated new possible options and tried to optimize the turnaround time for jobs. The improvement in the mean turnaround time for both active regional centers during this learning procedure is presented in Fig. 3.

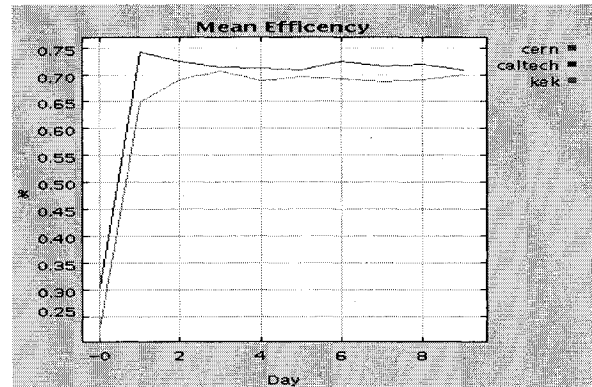


Figure 3. The improvement of the mean job efficiency (CPU time/ total time, during several days of learning.

SUMMARY

Using such a scheduling approach on quite simple problems in a realistic environment provided by the MONARC simulation tool seems to offer a possible solution for an efficient job distribution between Regional Centers. The difficult part in this approach is not the learning part but the decision making scheme which needs to effectively explore the “unknown” parts of the parameter space. Compared with a “classical” model we expect that such an approach may offer a better way to analyze the possible options and can evolve and improve itself dynamically.

REFERENCES

1. I.C. Legrand, H.B. Newman, “The Monarc Toolset for Simulating Large Network-Distributed processing Systems”, Proc. of the 2000 Winter Simulation Conference. And MONARC Simulation Tool http://www.cern.ch/MONARC/sim_tool/
2. B.Fritzke, “A self-organizing network that can follow non-stationary distributions, “ Proc. of the International Conference on Artificial Neural Networks '97, Springer, 1997, pp. 613-618.
3. M. Norgaard, O.Ravan, N.K. Poulsen and L.K. Hansen, Neural Networks for Modelling and Control of Dynamic Systems, Springer, 2000.