Soft Computing
Job Performance Evaluation Using Back-propagation Network

Bc. Petr Stehlík <xstehl14@stud.fit.vutbr.cz>

1 Introduction

Every user of a supercomputer needs to know whether their submitted job finished successfully and performed well. So far these tedious tasks are usually performed manually using only the output of their program and over-simplified metrics such as job run time and utilizied resources.

The aim of this project is to create a back-propagation neural network which can classify a job run whether it run well or it was in some way suspicious of unwanted behaviour such as poor performance or execution failure.

The network itself is supplied with fine-granular metric data acquired via Examon framework[2] which was run on Galileo supercomputer located in CINECA, Bologna, Italy. Where all job and metric data were gathered.

The structure of this document is as follows: in section 2 the theoretical background needed for this project is presented together with the description of Examon framework. In the following section 3 the data supplied to the network are described as well as the final format of the data. Afterwards in section 4 the implementation of the network and its structure are laid out and in the last two sections 5 and 6 the achieved results, summary and further work are discussed.

2 Theoretical Background

2.1 Backpropagation Network

Backpropagation networks are multi-layer feed-forward networks with supervised learning. There is no interconnection between neurons in the same layer but layers are fully connected to the next neighbouring layer in order to be able to do forward and backward propagation of values.

2.1.1 Forward-propagation

Each neuron in all layers but input layer disposes of a weight for each input initialized to a random value in range $< 0, 1 >$, linear base function and sigmoidal activation function.

When forward propagating an input vector the vector is laid out onto the input neurons and the vector is recalculated using following formulas for the next layer until the input vector is propagated to the output layer which outputs the response of the whole network itself.

The base function is shown in 1:

$$f(\vec{x}) = \sum_{i=0}^{n} w_i x_i \tag{1}$$

where $\vec{x}$ is the input vector, w are weights for each input of given neuron and n is the length of the neuron input vector and bias (term used as in [3][7]) value resulting in $n = |x| + 1$.

The sigmoidal activation function is presented in 2 where λ is a constant set to $\lambda = 1$.

$$g(u) = \frac{1}{1 + e^{-\lambda u}} \tag{2}$$

The output of a neuron is then given by using the equations 1 and 2:

$$y = g(f(\vec{x})) \tag{3}$$

2.1.2 Back-propagation

Back-propagation is used only when one trains the network and is one of the base methods for training feed-forward networks. The method is based on adjusting weights depending on the error calcuted by 4 for an output neuron p using produced output(o) and desired output (d) values.

$$E_p = \frac{1}{2}\sum_{j=1}^{m}(d_{pj} - o_{pj})^2 \tag{4}$$

The change of weights is calcuted using equation 5 where ∇E_p is the derived error gradient and μ the learning rate.

$$\Delta\vec{w_p} = -\mu\nabla E_p \tag{5}$$

To calculate one particular change of weight we use formula 6 where L is the given layer of the network, j is the j-th neuron in layer L and i is the i-th input of the neuron j.

$$\Delta w_{ji}^L = \mu\delta_j^L x_i^L \tag{6}$$

For the output layer L of the network we use formula 7.

$$\delta_j^L = (d_j - y_j^L)y_j^L(1 - y_j^L) \tag{7}$$

For hidden layers of the network we use the same principle propagating the error backward from the output layer as shown in equation 8 where $\lambda = 1$ and therefore left out.

$$\Delta^l w_{ji} = \mu^l\delta_j{}^l x_i = \mu\sum_{k=1}^{n_{l+1}}({}^{l+1}\delta_k{}^{l+1}w_{kj})^l y_j(1 -^l y_j)^l x_i \tag{8}$$

Each repetition of forward and backward propagation of all inputs is called an epoch. Finally we can choose when to update the weights, in batches after all input vectors were processed or using stochastic method where weights are updated after processing each input.

2.2 Examon

Examon[2] framework is used for exascale monitoring of supercomputing facilities. It is built on top of MQTT protocol[6] which allows measured metrics to be send to a central broker where received data are processed and stored in KairosDB[1] database utilizing Cassandra[5] cluster.

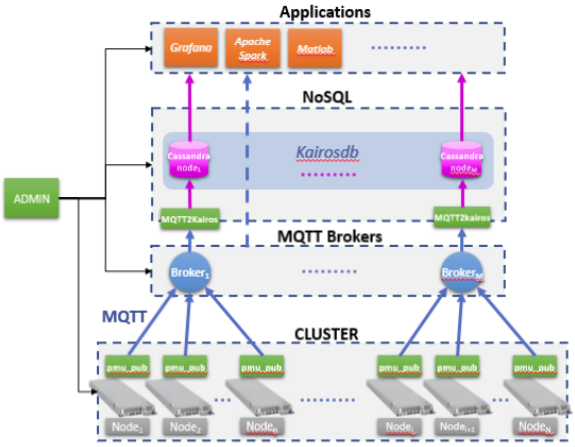

Figure 1: Examon framework architecture

KairosDB is used for storing metric data in time-series format whereas Cassandra, serving as a backend for KairosDB is also used for storing job-related data. More on data semantics is described in 3.

3 Data

As previously stated in 2.2 data are gather via Examon framework and stored in Cassandra database. We can split the data into two categories. Job data and metric data.

The job data come from PBSPro hooks which report various info about the job, mainly the allocated nodes, cores and other computational resources. The timestamps and events which are triggered by PBSPro during the lifecycle of the job. We use this data for determining the runtime of a job, its resource allocations and location of the job in a cluster (node names and core numbers). The job data are sent via MQTT and stored directly to a Cassandra cluster.

Using the job data one can query metric data. These are measured independently of job data and are monitored on per-core, per-CPU or per-node basis categorized into metrics. Each measured value is then sent via MQTT to KairosDB where it is stored according to its cluster location and metric. There are over 30 metrics monitored including but not limited to core load; C6 and C3 CPU state shares; system, CPU, IO and memory utilization or various temperatures gathered from various places in a node.

For the project, twelve major metrics were chosen for the best reflection of job performance. A short summary of the chosen metrics is shown in table 1.

Metric name	Metric tag	unit	sampling rate	base
core's load	load_core	%	2s	per-core
C6 states	C6res	%	2s	per-core
C3 states	C3res	%	2s	per-core
instructions per second	ips	IPS	2s	per-core
system utilization	Sys_Utilization	%	20s	per-node
CPU utilization	CPU_Utilization	%	20s	per-node
IO utilization	IO_Utilization	%	20s	per-node
memory utilization	Memory_Utilization	%	20s	per-node
L1 and L2 bounds	L1L2_Bound	%	2s	per-core
L3 bounds	L3_Bound	%	2s	per-core
front-end bounds	front_end_bound	%	2s	per-core
back-end bounds	back_end_bound	%	2s	per-core

Table 1: Overview of measured metrics

KairosDB provides us with a REST API for querying metric data in various ways. The queries are formed using JSON objects and results are also returned as JSON objects. The KairosDB limits all stored data to 21 days

For complete data acquisition we combined the job data and metric data together and queried only jobs which fit several conditions described in 4.1.

4 Implementation

In this section we describe the implementation phase of the project. The project was implemented using Python 2.7.13. External library SciPy[4] was used for linear data interpolation.

4.1 Data Acquisition

First, job data were queried and filtered according to several rules:

- job run time must be between 10 and 60 minutes
- job must occupy the whole node (multiplies of 16 cores)
- job must be run in 15-day period

4.2 Data Labelling

4.3 Metric Networks

4.4 Job Network

5 Achieved Results

6 Summary

References

[1] KairosDB. `https://kairosdb.github.io`. Accessed: 2017-11-25.

[2] F. Beneventi, A. Bartolini, C. Cavazzoni, and L. Benini. Continuous learning of hpc infrastructure models using big data analytics and in-memory processing tools. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 1038–1043, March 2017.

[3] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[4] E. Jones, T. Oliphant, and P. Peterson. {SciPy}: open source scientific tools for {Python}. 2014.

[5] A. Lakshman and P. Malik. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, 2010.

[6] D. Locke. Mq telemetry transport (mqtt) v3. 1 protocol specification. *IBM developerWorks Technical Library*, 2010.

[7] K. Mehrotra, C. K. Mohan, and S. Ranka. *Elements of artificial neural networks*. MIT press, 1997.