



TASK 3.3 PROTOTYPE DOCUMENTATION

WP3.3 – Monitoring Infrastructure (ANALYSIS)

Document Filename:	CG3.3.4-v0.5-ICM-PrototypeDoc.doc
Work package:	WP3.3
Partner(s):	TDC,CYFRONET,ICM
Lead Partner:	TDC
Config ID:	CG-3.3.4-D3.3-v0.5-ICM010- PostProcessingAnalysis
Document classification:	PUBLIC

Abstract:

This document describes the first prototype of the application for post-processing analysis of Grid monitoring data.

Delivery Slip

	Name	Partner	Date	Signature
From				
Verified by				
Approved by				

Document Log

Version	Date	Summary of changes	Author
0.5	16/01/2002	Draft version	J.Jurkiewicz, K.Nawrocki, A.Padee, A.Wislowski

CONTENTS

EXECUTIVE SUMMARY	4
1. INTRODUCTION	5
1.1. PURPOSE	5
1.2. DEFINITIONS, ABBREVIATIONS, ACRONYMS	5
2. REFERENCES	6
2.1. SOURCE CODE	6
2.2. CONTACT INFORMATION	6
3. IMPLEMENTATION STRUCTURE	7
3.1. THE DATA INPUT MODULE	7
3.2. THE FORECASTER MODULE	8
3.3. THE VIEWER MODULE	9
4. PROTOTYPE FUNCTIONALITY	10
4.1. INPUT INTERFACE	10
4.2. OUTPUT INTERFACE	10
4.3. PROCESSED PARAMETERS	10
4.4. AUTOMATION	10
5. USER MANUAL	12
5.1. DEPENDENCIES	12
5.2. INSTALLATION	12
5.3. RUNNING	13
5.4. INTERFACE DESCRIPTION	13
6. INTERNAL TESTS	14
6.1. PERIODIC USAGE PATTERN	14
6.2. RANDOM USAGE PATTERN	17
6.3. CONSTANT USAGE PATTERN	19

EXECUTIVE SUMMARY

This document provides description of the first prototype of the application for post-processing analysis of Grid monitoring data. It gives overview of the structure of the prototype, compares its functionality to the functionality of the final application, provides practical instructions how to install and use the prototype and finally describes results of the test that were performed to estimate behaviour the of the prototype under typical computer usage conditions.

1. INTRODUCTION

1.1. PURPOSE

The goal of the Post-processing Analysis monitoring application is to collect and analyse data provided by other CrossGrid monitoring subtasks, i.e. the Jiro-based monitoring tool and SANTA-G monitoring tool (described in their own Software Requirements Specifications). Our application have to provide SQL-query-based output of the monitoring data, which will be utilized by the Performance Prediction software (task 2.4.2).

As a post-processing tool the application described in this document should have access to the data provided by first prototypes of other monitoring tasks which is unfeasible because the dates of all monitoring tasks deadlines for publishing its first prototypes are the same. So the main problem we had to face was lack of sources of data that our application will be provided in the future. To overcome this problem we decided to make an 'ad-hoc' solution by writing our own, very simple data producer working on a local node. As a consequence our first prototype is a very limited version of the final application. Main purpose of the prototype was to construct and test a framework for future development based on real data sources.

1.2. DEFINITIONS, ABBREVIATIONS, ACRONYMS

AS	Analysis System
bash	Bourne Again Shell
CrossGrid	the EU CrossGrid Project IST-2001-32243
GA	Genetic Algorithm
GUI	Graphical User Interface
IS	Information System
JDBC	Java DataBase Connector
JOONE	Java Object Oriented Neural Engine
NN	Neural Network
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SRS	Software Requirements Language
Task3.3.4-SRS	Task 3.3.4 Post-processing Analysis Design Document

2. REFERENCES

2.1. SOURCE CODE

Subtask 3.3.4 is fully open source project. The whole source code can be downloaded from Crossgrid CVS repository at FZK (Karlsruhe, Germany):

http://gridportal.fzk.de/cgi-bin/viewcvs.cgi/crossgrid/crossgrid/wp3/wp3_3-moninfr/wp3_3_4-postproc/

Sources of the Post-Processing system itself are located in the directory AS/as, external tools necessary to run the application in the directory AS/ext, and the documentation in the directory AS/doc.

2.2. CONTACT INFORMATION

Contact information for partners responsible for the prototype and for technical staff:

Krzysztof.Nawrocki Krzysztof.Nawrocki@fuw.edu.pl

3. IMPLEMENTATION STRUCTURE

The architecture of the Post-Processing Analysis task can be found in the [Task 3.3.4 Post-processing Analysis Design Document](#). In this chapter the structure of the first prototype is given.

3.1. THE DATA INPUT MODULE - SYSSTAT

As stated earlier in this document one of the main problem to solve when building the prototype was lack of monitoring data which in the future will be provided by other monitoring tasks. We decided to solve the problem by building a very simple Data Input Module called sysstat as a temporary replacement for the GridConnector. We want to stress that this is an ‘ad-hoc’ solution. Its functionality is very limited with respect to the future functionality of the GridConnector. The gathered data are limited only to local host and are based on the results provided by the UNIX ‘top’ command. The monitored system parameters are written to the MySQL database which corresponds to the future ASDb module.

The monitored parameters are (names are self-explanatory):

```
mysql> describe sysstat;
```

Field	Type	Null	Key	Default	Extra
forecast	tinyint(1)		PRI	0	
hostname	varchar(30)		PRI		
unixtime	bigint(20)		PRI	0	
unixtime_forecast	bigint(20)	YES		NULL	
date	date	YES		NULL	
time	time	YES		NULL	
top_time	varchar(10)	YES		NULL	
uptime	varchar(30)	YES		NULL	
nusers	int(11)	YES		NULL	
load_av1m	double	YES		NULL	
load_av5m	double	YES		NULL	
load_av15m	double	YES		NULL	
nprocesses	int(11)	YES		NULL	
nprocesses_sleeping	int(11)	YES		NULL	
nprocesses_running	int(11)	YES		NULL	
nprocesses_zombie	int(11)	YES		NULL	
cpu_user	double	YES		NULL	
cpu_system	double	YES		NULL	
cpu_nice	double	YES		NULL	
cpu_idle	double	YES		NULL	
mem_av	bigint(20)	YES		NULL	
mem_used	bigint(20)	YES		NULL	
mem_free	bigint(20)	YES		NULL	
mem_shrd	bigint(20)	YES		NULL	

```

| mem_buff          | bigint(20) | YES | | NULL | | |
| swap_av           | bigint(20) | YES | | NULL | | |
| swap_used         | bigint(20) | YES | | NULL | | |
| swap_free         | bigint(20) | YES | | NULL | | |
| swap_cached       | bigint(20) | YES | | NULL | | |
| ncpus             | int(11)    | YES | | NULL | | |
| cpu0_user         | double    | YES | | NULL | | |
| cpu0_system       | double    | YES | | NULL | | |
| cpu0_nice         | double    | YES | | NULL | | |
| cpu0_idle         | double    | YES | | NULL | | |
| cpu1_user         | double    | YES | | NULL | | |
| cpu1_system       | double    | YES | | NULL | | |
| cpu1_nice         | double    | YES | | NULL | | |
| cpu1_idle         | double    | YES | | NULL | | |
+-----+-----+-----+-----+-----+-----+
38 rows in set (0.00 sec)

```

The sysstat is a Python script using MySQL-Python library to access the ASDb.

3.2. THE FORECASTER MODULE

The aim of this prototype was to build a working mechanism for generation of the forecasts. In this state of work we do not intend to produce exact forecasts, but to present the framework for the application.

The first Forecaster we made is the NNForecaster that is intended to predict the future state of Grid basing on the its usage in periods of time, as days, weeks, months and years (the first prototype for easiness of providing NNForecaster tests uses time expressed in minutes and seconds only). Its architecture was based on the neural network described below.

The input to the neural network is the specification of the time and the output is the specification of the parameters of the GridState. So far we have used as the output only a small subset of the parameters provided by sysstat. Also the input is simplified. In the final version the input will consist of date and time description (month, day of the month, hour, minutes).

Below set of input and output parameters is presented:

- Input description:
 - seconds
 - minutes

- Output description:
 - cpu used by user
 - cpu used by system
 - cpu idle
 - memory used

Three layers neural network was used and neurons were coded in the zero-one model. Input parameters were coded in 14 neurons, and output in 15 neurons. Hidden layer used 30 neurons. Every GridState parameter was coded as percent of possible result. The preciseness was 3 percent.

To conclude we built a framework for the Forecaster, which in the Design Document was one of the two parts of NNForecaster. In the next step we intend to do is to configure the neural network we used to work better. In this place it is important to actualize new knowledge faster and more precisely. To make neural network work better we plan to implement genetic algorithms for pre-processing. So in place of building new GAForecaster we will use this mechanism in the NNForecaster. In the future we will build forecasters for time series analysis, named in the Design Document TSForecaster and HAForecaster which will also verify forecasts made by all of the forecasters.

Tests we made aimed to present that our framework works. We were able to make forecasts basing on a little knowledge.

The Forecaster module uses the JOONE application, which provides a library to implement neural networks in Java.

3.3. THE VIEWER MODULE

To present parameters monitored by sysstat and parameters forecasted by the Forecaster we provide our prototype with the module which enables a user to make simple visualization of results.

In the final version of our application this goal will be achieved using Grid Portals and the data will be available thru the WWW for authorised users. Now, as this functionality of Grid Portal is not available, in the first prototype a user can view data as JPEG images generated on request.

In this prototype we use free software: Gnuplot, SQL-queries to the ASDb and bash to integrate it. The structure is simple - in the loop program reads which parameters are to be plotted from the configuration file, then asks ASDb for their values, generates script for Gnuplot and finally changes data format from png to jpeg.

This structure is very simple, but turned out to be very efficient. As stated above in the next versions of our software this approach will be replaced by a framework compatible with Grid Portals technology.

4. PROTOTYPE FUNCTIONALITY

For the needs of the first prototype functionality of the application has been limited in several points. Although overall architecture of the system is similar to the final version, the main differences occur on the input/output interfaces. The reason for this, as stated before, is absence of other monitoring modules, on which Post Processing application strongly depends.

4.1. INPUT INTERFACE

Current version reads data via it's own module named Sysstat. This operation can be done only through shared filesystem, so is limited to clusters or AFS-enabled grid nodes. In the future releases this module will be replaced by the GridConnector module, which will be able to read data via SOAP (from Jiro-monitoring system), and R-GMA (from SANTA-G).

4.2. OUTPUT INTERFACE

In the prototype release output is provided only via Viewer module, which is able to provide static images in JPEG format (though it is also possible to read output directly from the database). In the future release Viewer module will be equipped with Web portals connectivity, thus able to display post-processed data on-line.

As the main consumer of post-processed monitoring data inside Crossgrid middleware will be Task 2.4.2 (Performance Prediction), in the final release output will be provided by the GridConnector module via R-GMA interface.

4.3. PROCESSED PARAMETERS

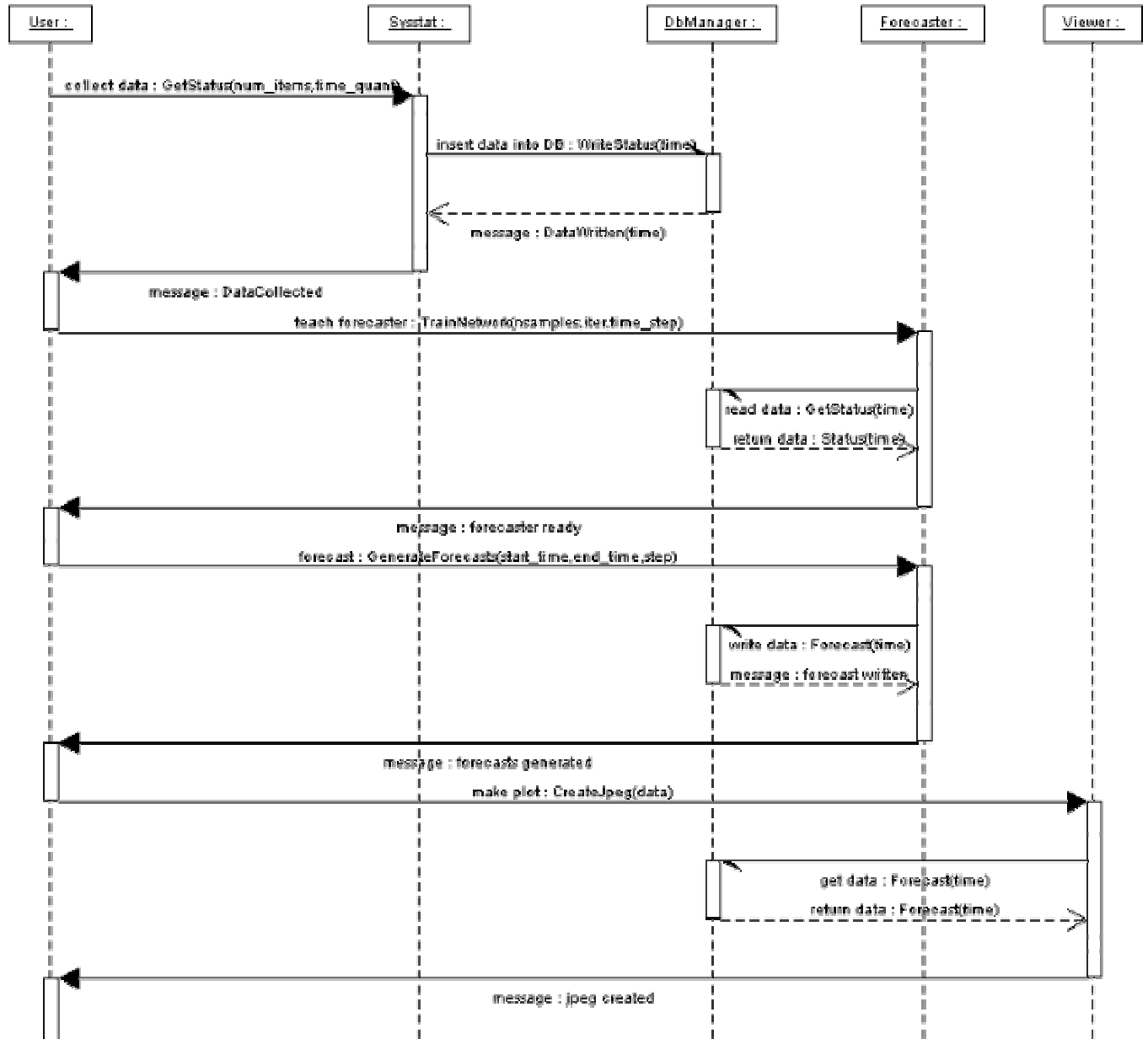
Prototype version processes only a small subset of the available parameters. For now, this includes data like CPU usage (user and overall), and the memory usage. In the final release new sources of information will allow to extend this set by many cluster and network related parameters.

4.4. AUTOMATION

For the test purposes, prototype version requires much more user interaction than the future release. The main difference is that the software now works in "passive" mode, which means that data collection process must be explicitly triggered by user (in other words, the application doesn't acquire the historical data about cluster usage by itself). This is very convenient for the test purposes, because tester can generate regular patterns on the desired parameter and then start the monitoring. Disadvantage of this approach is that the forecast cannot be generated ad hoc.

Apart from that, forecaster's learning cycles are also triggered by hand, and it must be done before the forecasting takes place. In the final version all these preparations will be hidden from user's point of view, and the application will be constantly ready for forecasts generation.

All these issues are illustrated by the sequence diagram showed below:



5. USER MANUAL

5.1. DEPENDENCIES

- Prerequisites

All software components listed below are available as RPMS in the EDG Globus distribution:

- [MySQL](#) i [MySQL-Python](#)

- MySQL-python-0.9.2-1
- MySQL-client-3.23.42-1
- MySQL-shared-3.23.42-1
- MySQL-devel-3.23.42-1
- MySQL-3.23.42-1

- [Python](#)

- python-1.5.2-27.6.x

- [Java](#)

- j2sdk-1.4.0-fcs

- [Gnuplot](#)

- gnuplot-3.7.1-5

- External software distributed with the prototype:

- [JOONE](#)

- joone-engine.jar version 0.9.8

- [MySQL Connector/J2](#)

- mysql-connector-java-2.0.14-bin.jar

- pngtojpeg

- UNIX 'top' command: version for 2 processors

- [qsbench](#)

5.2. INSTALLATION

1. All RPMS listed in section 5.1 as prerequisites have to be present on the host on which the AS software is to be installed
2. the latest version of the AS distribution AS-0.1.latest.tgz has to be downloaded from the CrossGrid CVS server: http://gridportal.fzk.de/cgi-bin/viewcvs.cgi/crossgrid/crossgrid/wp3/wp3_3-moninfr/wp3_3_4-postproc/download/AS-0.1.latest.tgz
3. the package AS-0.1.latest.tgz has to be untarred and configuration script has to be executed from the directory containing AS scripts and executables:

```
>tar -xvzf AS-0.1.latest.tgz
```

```
>cd AS-0.1/as/bin  
>./as_conf.sh
```

5.3. RUNNING

When the application and the external software described in Section 5.1 are installed one can use the AS prototype following the steps listed below (\$BASE is the directory where the file AS-0.1.latest.tgz has been untarred):

- Collect the data:
 1. go to the configuration directory: `cd $BASE/AS-0.1/as/etc`
 2. define benchmark parameters in the file `qsbench.cnf`
 3. define sysstat parameters in the file `sysstat.cnf`
 4. run `$BASE/as/bin/qsbench.run` and `$BASE/AS-0.1/as/bin/sysstat.run` at the same time, ex.:

```
>cd $BASE/AS-0.1/as/bin  
>sysstat.run &; qsbench.run
```
- Train the neural network:
 5. `>$BASE/AS-0.1/as/bin/Forecaster.sh <start_time_learning> <endtime_learning>\
<number_of_probes_to_learn> <number_of_cycles_of_learn>`
- Make the forecast:
 6. `>$BASE/AS-0.1/as/bin/do_forecast <forecast_start_time> <forecast_endtime> <step>`
- Generate pictures:
 7. define plots parameters in the file `$BASE/AS-0.1/as/etc/plot.cnf` and create file `$BASE/AS-0.1/as/etc/monitored_parameters_<MySQL_table_name>` according to the instructions given in `$BASE/as/etc/plot.cnf`
 8. `>$BASE/AS-0.1/as/bin/plot.sh <start_time> <end_time> <zerotime>`
zerotime sets zero on the time axis. Time on the axis is expressed in seconds.

5.4. INTERFACE DESCRIPTION

This prototype does not provide any API.

6. INTERNAL TESTS

The prototype has been tested on the data generated on a computer being a Working Node of the ICM CrossGrid cluster (xgrid4.icm.edu.pl).

It is a 2 x Pentium III, 1GHz machine, equipped with 917080 kB RAM and running Red Hat Linux with kernel 2.2.19-6.2.16smp.

The purpose of the tests concentrated on testing the prototype of the NNForecaster. The tests procedure consisted of executing sequence of commands described in Section 5.3 for different load and memory usage patterns.

The main problem in making tests was lack of real data. We were generating them using qsbench benchmarks.

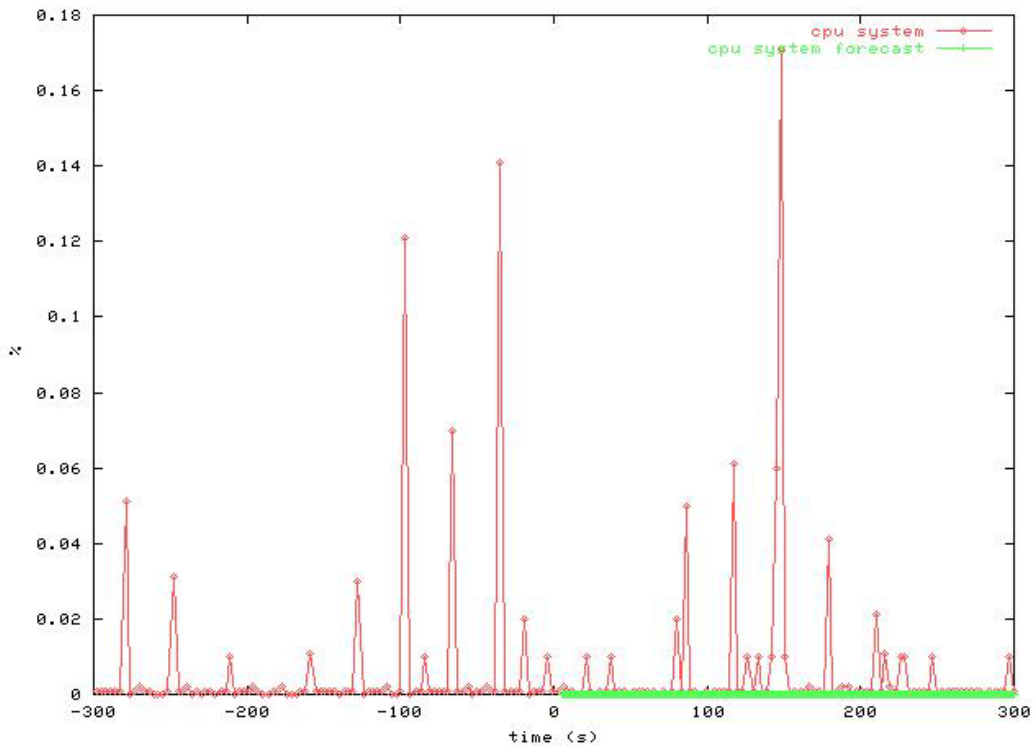
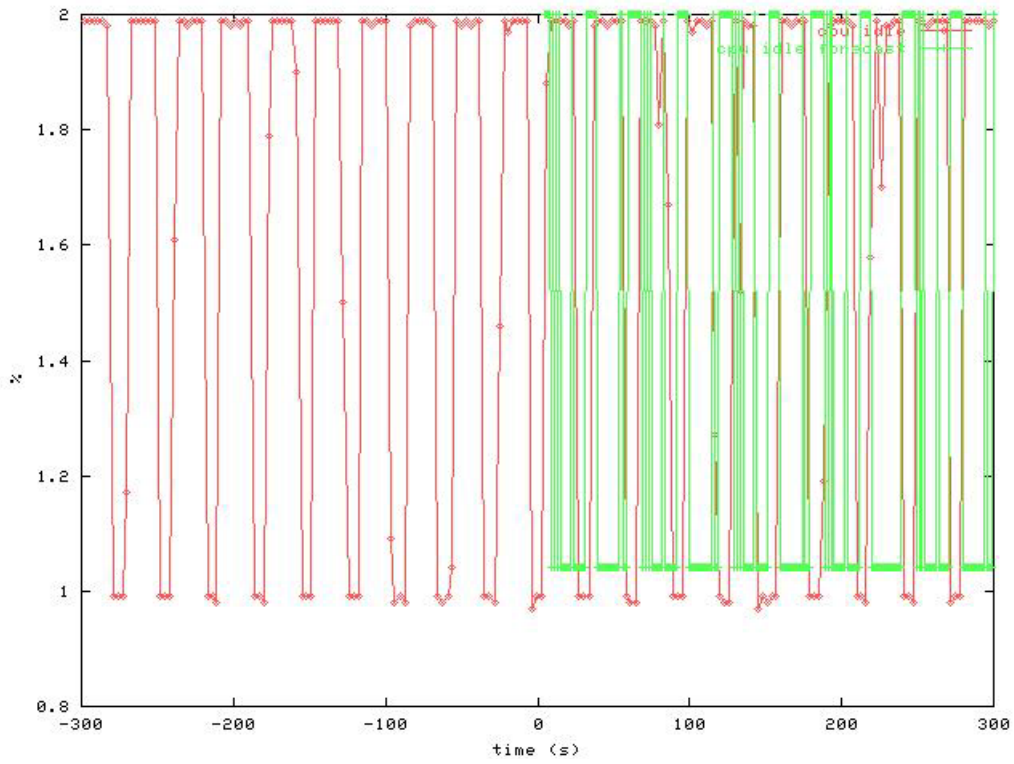
We have decided to use three styles of usage patterns.

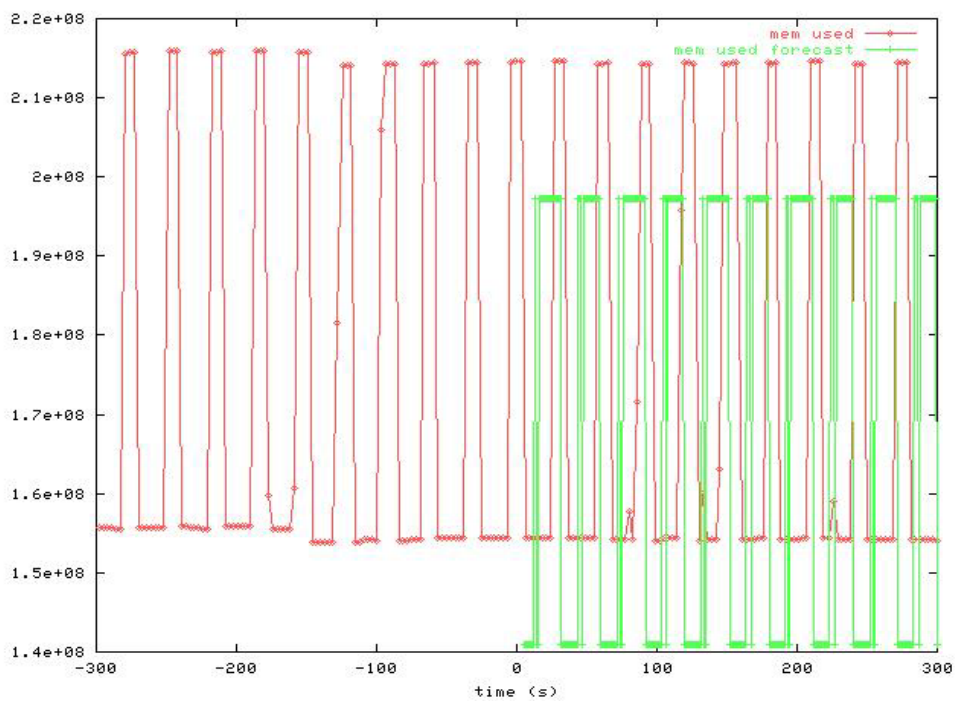
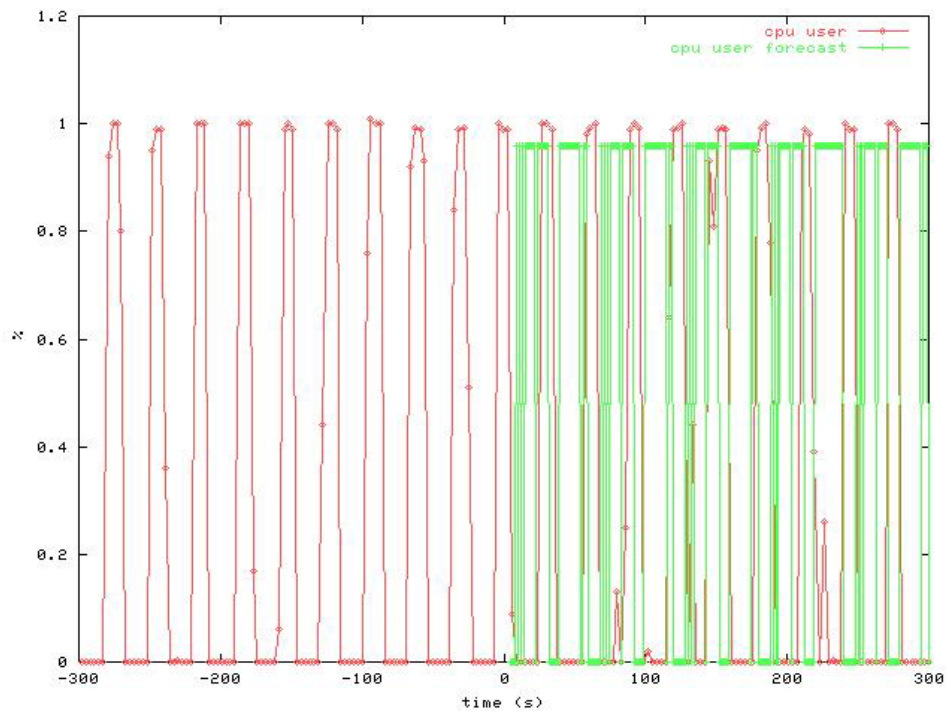
The results of the tests are presented below. On plots red color describes real data and green color presents predictions of the NNForecaster. For each pattern four parameters are presented: cpu_idle, cpu_system, cpu_user and mem_used.

6.1. PERIODIC USAGE PATTERN

Data are generated by running benchmark once per 30 second. Program runs for 10 seconds and then sleeps for 20 seconds.

As you can see on pictures, our model is not brilliant, but good enough for finding some cycles. As one can see the Forecaster predictions are shifted with respect to real data. Other problem is wrong normalization of memory usage (mem_used). The mistake can be caused by rounding of parameters. The relative accuracy of predicted parameters is equal 3%. It probably means that we have to change parameters resolution in next version of the Forecaster.

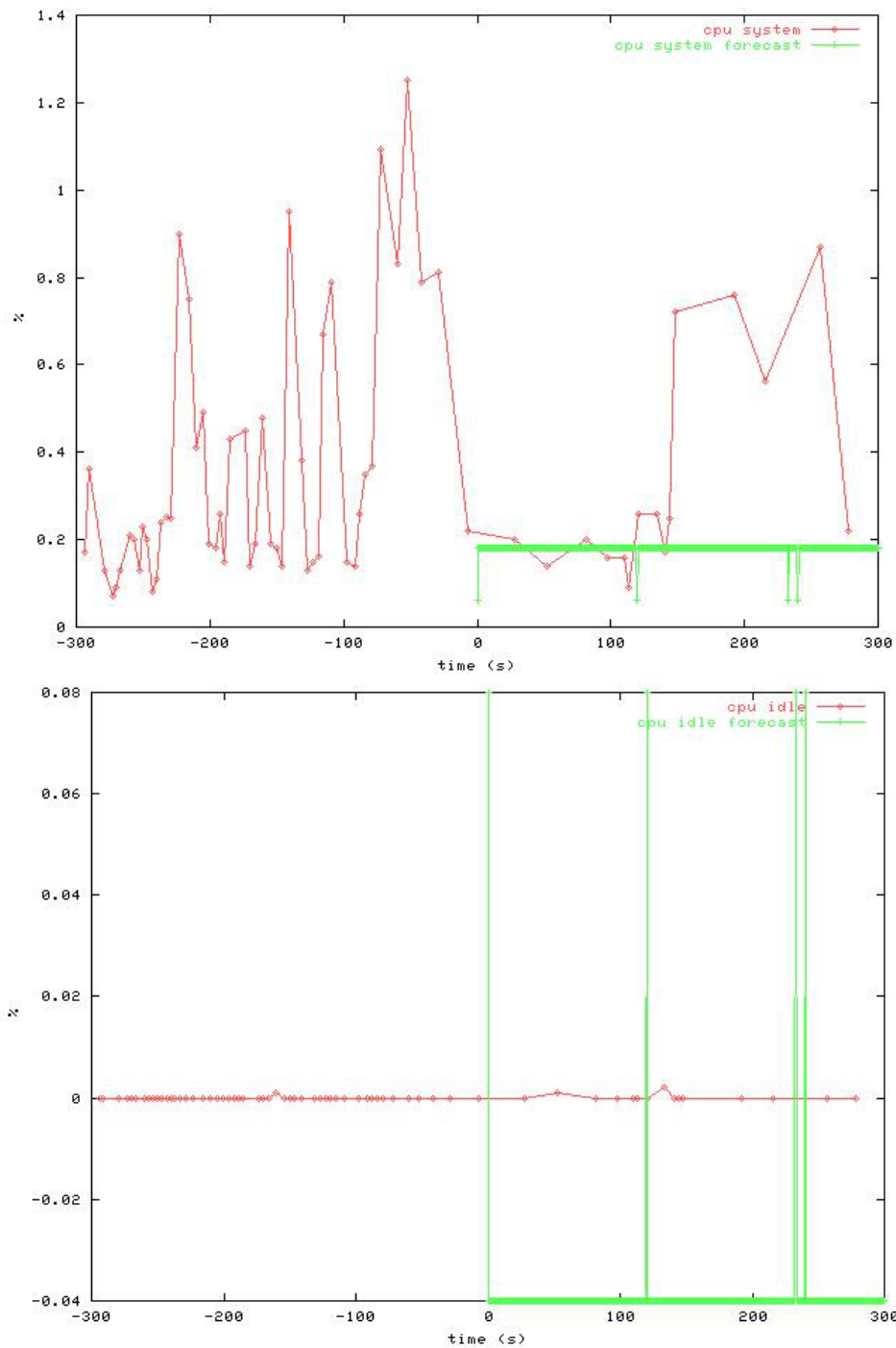


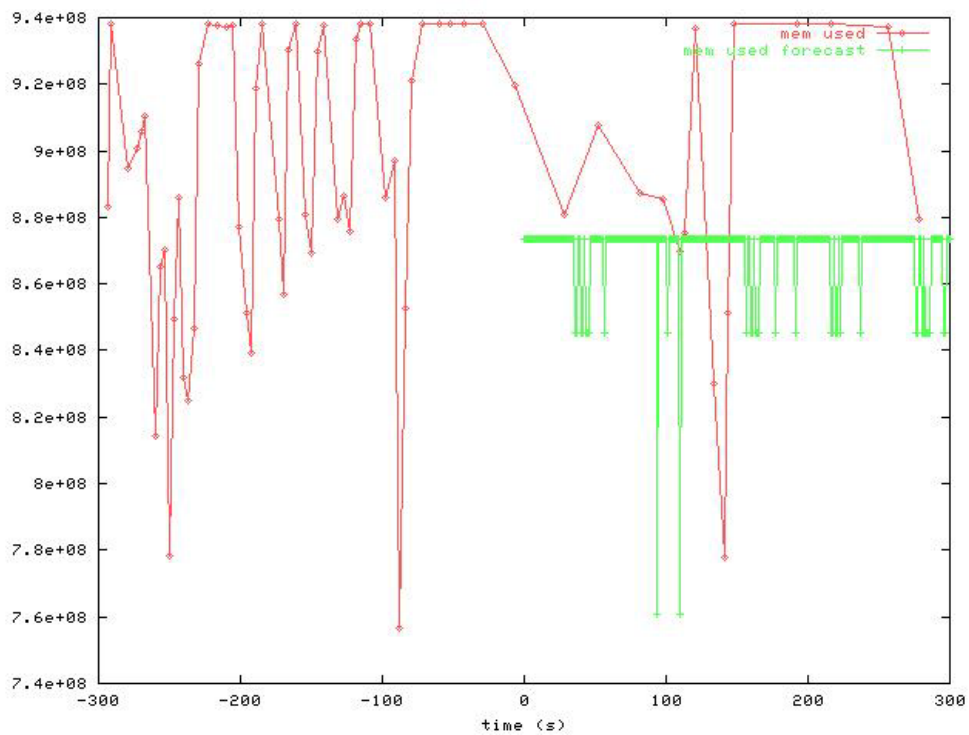
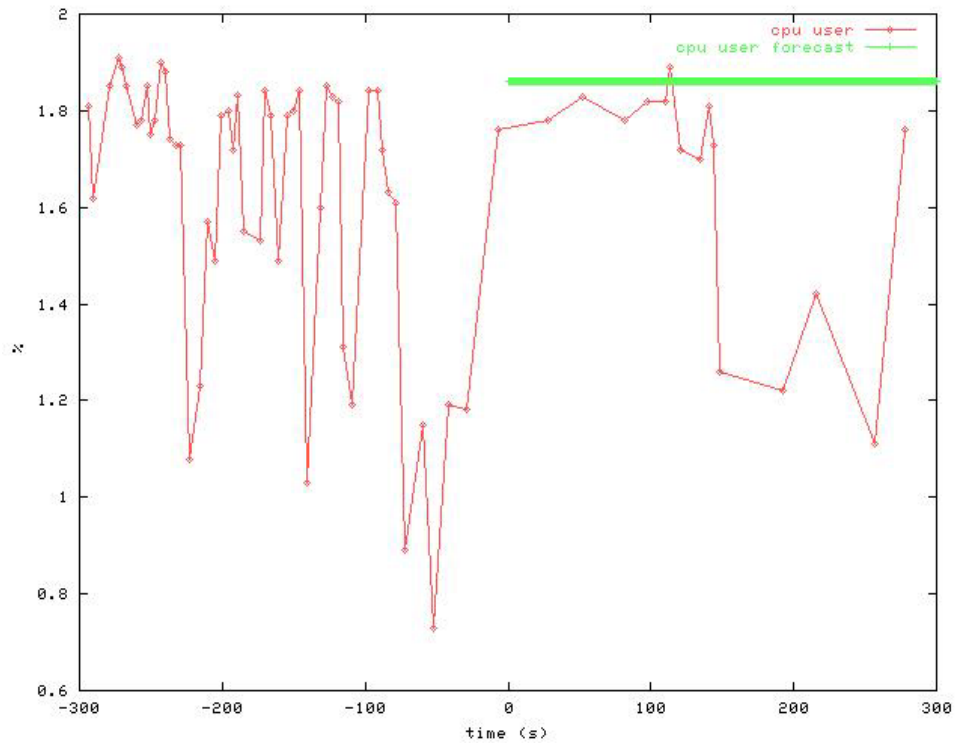


6.2. RANDOM USAGE PATTERN

Many benchmarks were run in parallel using a lot of memory and have some problems with contexts changes

We have made two forecasts for this model, using small and big number of learning cycles. For small number of learning cycles the results are very unstable (not presented here). When the number of cycles is big our network just averages the input data.





6.3. CONSTANT USAGE PATTERN

Program runs all time using constant memory and constant CPU (200%) with little breaks for re-swapping.

The parameters are not as constant as one could suppose, because sometimes, the system has to perform some activity. This mistakes our Forecaster also in this case. This means that the performance of the Forecaster has to be improved.

