DISTRIBUTED COMPUTING IN DATA PROCESSING

A. Pavelka and A. Procházka
Institute of Chemical Technology, Department of Computing and Control Engineering

Abstract

The paper describes basic operations associated with the use of the distributed computing toolbox and its application for processing of extensive and complex mathematical problems using the computer network and the set of computers for parallel processing of separate components of the whole algorithm.

1 Introduction

There is a wide range of problems and applications calculation of which performed on a single computer takes too much time. To speed up such calculations and effectively use all computer power around it is suitable to use distributed computing. MATLAB offers one elegant internal solution of this problem.

2 Principles

For time consuming problems we have possibility to separate the whole calculation into smaller elements and process them separately. In the MATLAB environment we can provide it using the Distributed Computing Toolbox (DCT). This system is important for communication between the user's computer and the calculation cluster created by the MATLAB Distributed Computing Engine (DCE). The DCT and the MATLAB DCE enable us to coordinate and execute independent MATLAB operations simultaneously on a cluster of computers, speeding up execution of large MATLAB jobs [1, 2].

In the terminology of DCT and DCE the *job* represents some large operation that we need to perform in the MATLAB session. A *job* is divided into small elements called *tasks*. Every *task* is evaluated by a *worker*. A *worker* is a calculation session, it could be a single computer or it is possible to have more *workers* on one computer. All *jobs* and *tasks* are managed by the head node controlling computer called the *job manager*.

The optional *job manager* can run on any machine on the network. The *job manager* runs *jobs* according to Fig. 1 in the order in which they are submitted, unless any *jobs* in its queue are promoted, demoted, canceled, or destroyed.

Each *worker* receives a *task* of the running *job* from the *job manager*, executes the *task*, returns the result to the *job manager*, and then receives another *task*. When all *tasks* for a running *job* have been assigned to *workers*, the *job manager* starts running the next *job* with the next available *worker*.

A MATLAB DCE setup usually includes many *workers* that can execute *tasks* simultaneously, speeding up execution of large MATLAB *jobs*. It is generally not important which *worker* executes a specific *task*. Each *worker* evaluates *tasks* once a time, returning results to the *job manager*. The *job manager* then returns results of all *tasks* in the *job* to the client session.

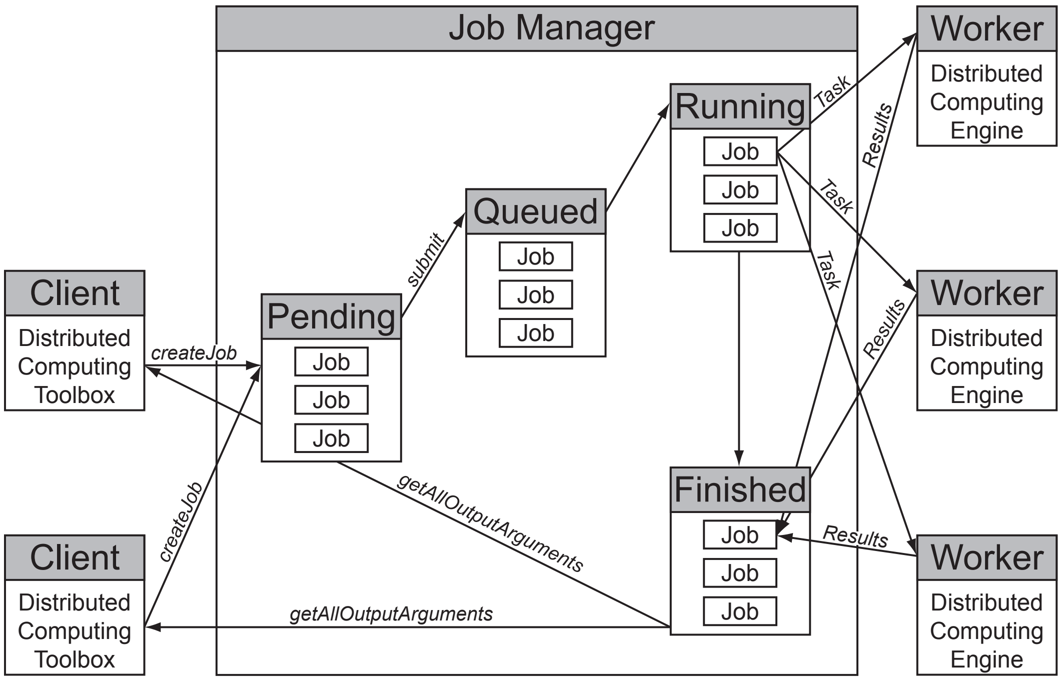


Figure 1: The priciples and architecture of the MATLAB Distributed Computing Toolbox and the Distributed Computing Engine

3 Programming Methods for Distributed Computing

A typical source code for the DCT must contain the following steps:

1. Find a Job Manager
2. Create a Job
3. Create Tasks
4. Submit a Job to the Job Queue in Job Manager
5. Retrieve the Jobs Results (wait until the job is finished)
6. Destroy the Job

The whole process can be illustrated by a simple example. Let's have a measured data set (values x and y) and model them by relation $y = y_0\,(1 + (a-1)\,y_0^{a-1}\,b\,x)^{1/(1-a)}$ that describe measured data analytically. The problem is in the design of algorithm for solution of the classical problem of estimating suitable parameters for measured data and the selected model.

First of all we define the function. This function will be used for calculation of values y from measured vales x using the searched parameters a and b.

```
function y = my_function(y0,a,b,x)
warning off
y = y0*(1+(a-1)*y0.^(a-1)*b*x).^(1/(1-a));
```

In the next step we define function that will represent one *task* of calculations. This function `my_Task` is the key function as it will be evaluated on *workers*, so it should be relatively complicated and time consuming function. The selected example of raw calculation or so called brutal force method is an ideal example for demonstration of DCT because it always leads to the exact and perfect solution. But its biggest disadvantage is in the enormous time, that is necessary to have for its calculation. In other words the parameters of `my_Task` are two vectors A and B, that represent range in which the unknown parameter will be searched by repeated evaluation of `my_function`. Outputs of the *task* function include the minimal error, parameters corresponding with this minimal error, the whole error matrix and calculation time.

```
function [e_min,parameter_a,parameter_b,error,calculation_time] = my_Task(A,B)

%% Data load a definition
load my_data

x0 = x(1);   x = x(2:end);
y0 = y(1);   y = y(2:end);

%% Own Calculation
error = zeros(length(A),length(B));
tic
for i = 1:length(A)
    for j = 1:length(B)
        C  = my_function(y0,A(i),B(j),x);
        error(i,j) = sum(C-y).^2;
    end
end
calculation_time = toc;

%% Error, Parameters and Limitation of Error
error(error>0.5) = nan;
e_min = min(error(:));
[xm,ym] = find(error==e_min);
parameter_a = A(xm(1));
parameter_b = B(ym(1));
```

The following script includes the main steps that are necessary to perform before calculations using the DCT and DCE. In the initial part of the script there are standard definitions of parameters and clearing workspace. The most important part starts by the comment `%% Parameter of DCT` on the line 18, it searches for the available *job manager* and creates the new empty *job* in the founded *job manager*. The next lines shows important parameters that are useful to always check and setup, it includes restarting MATLAB *workers* before evaluating *job tasks*. Property `FileDependencies` gives us the opportunity to define a list of all the directories and files that are needed for calculations. When this property is set to true or 1 directories and files are zipped and sent to the workers. At the worker side, the data is unzipped, and the entries defined in the property are added to the path of the MATLAB worker session. It is also possible to setup own specific name of the *job* that later helps in the *job* identification.

```
 1  %% Data load a definition
 2  close all; clear all; clc;
 3  load my_data
 4
 5  x0 = x(1);   x = x(2:end);
 6  y0 = y(1);   y = y(2:end);
 7
 8  %% Parameter definition
 9  A = 1.01:0.01:129; % range for parameter a
10  B = 0.01:0.01:128; % range for parameter b
11  k = 32; % number of intervals for separating of A
12
13  %% Check of the calculation start
14  disp(['The calculation matrix is ',...
15      num2str(length(A)),'x',num2str(length(B)),'.'])
16  disp(['     i.e. ',num2str(length(A)*length(B)),' combinations.'])
17
18  %% Parameter of DCT
19          jm = findResource('jobmanager');
20          job = createJob(jm);
21          set(job,'RestartWorker',0);
22
23          dir_m  =  dir('my_*.m');
24          dir_mat = dir('my_*.mat');
25          dir_cell = struct2cell([dir_m; dir_mat]);
26          dir_cell = dir_cell(1,:);
27          set(job,'FileDependencies',dir_cell)
28          set(job,'Name',['My Calculation - ',getenv('COMPUTERNAME')]);
29
30  %% Own calculation
31  for i = 1:k
32      step = length(A)/k;
33      A_task = A(step*(i-1)+1:step*i);
34      createTask(job, @my_Task, 5, {A_task,B},'CaptureCommandWindowOutput',1);
35  end
36  submit(job);
37  disp('done')
```

The final section of the script starting with comment `%% Own calculation` on the line 30 separates the wide range of parameter A into k smaller intervals. Function `createTask` creates *task* to `job` that now represents the handle to the just creating *job*. This task will be evaluating function `my_task` with parameters `(A_task,B)` expecting 5 output parameters and during evaluation an output produced by function `my_task` in the MATLAB *Command Window* will be gathered. The line

```
  createTask(job, @my_Task,5,{A_task,B},'CaptureCommandWindowOutput',1);
```

is equal to

```
  [e_min,parameter_a,parameter_b,error,calculation_time] =  my_Task(A_task,B)
```

in the language of non-distributed programming, of course that this comparison does not include storing of output of the `my_Task` function in the `for - end` cycle. The final command, `submit(job)`, sends the whole *job* into the *job manager*. After the evaluation of this script we passed 4 of 6 steps needed for every MATLAB distributed calculation.

When the *job* is in state `finished` you can provide following commands in the MATLAB *Command Window* to retrieve all data, save them and destorying (removing, cleaning) your finished *job* from the *job manager*

```
data = getAllOutputArguments(job);
save data32 data
destroy(job)
```

The result processing must be done after retrieving and saving results. This processing is usually not connected with the DCT or DCE and can be provided "off-line" (without presence of *job managers or workers*). In the result processing we have to be very careful to the RAM because sooner or later we create a large *job* with many *tasks* and the final output cell matrix will exceed the computer memory limitations. To avoid this problem there exists a solution providing saving results in every step and storing them on the safe place for example via FTP. Aternativelly we have possibility to process results of every *task* directly from the *job manager*. Those solutions will be described later.

From our code of the post-processing script we describe section 5 and 6 that is on lines 24–39 beginning with `%% DCT Results` and `%% Minimal Error Search` comments. The section 5 extract some of resulting outputs from the cell matrix `data` that have been created by `getAllOutputArguments` function. Section 6 directly selects desired error matrix with the lowest error from cell matrix `data`. Finally the `data` variable is cleared. This way of handling with the final cell matrix including calculated result is caused be the huge consumption of computer memory. Totaly matrix of size 12 800 $\times$ 12 800 has been investigated i.e. 163 840 000 combinations of parameters a and b that has been evaluated[1].

```
 1  %% Data load a definition
 2  close all; clear all; clc;
 3  load my_data
 4
 5  x0 = x(1);   x = x(2:end);
 6  y0 = y(1);   y = y(2:end);
 7
 8  %% Parameter definition
 9  A = 1.01:0.01:129; % range for parameter a
10  B = 0.01:0.01:128; % range for parameter b
11
12  %% Input data
13  f1=figure;
14      plot(x,y,'b','LineWidth',2)
15      grid on
16      xlabel('values x','FontWeight','bold')
17      ylabel('values y','FontWeight','bold')
18
19  %% Check of the calculation start
20  disp(['The calculation matrix is ',...
```

[1]The size of results are following: `data32.mat` 46.7 MB, variable `data` 1 250 MB, variable `ERROR` 39.1 MB

```
    num2str(length(A)),'x',num2str(length(B)),'.'])
disp(['     i.e. ',num2str(length(A)*length(B)),' combinations.'])

%% DCT Results
load data32
for i=1:size(data,1)
    disp(['Calculation cycle No: ',num2str(i)]);
    min_ERROR(i,1) = data{i,1};
    parameter_a(i,1) = data{i,2};
    parameter_b(i,1) = data{i,3};
    calculation_time(i,1) = data{i,5};
end

%% Minimal Error Search
    total_min = min(min_ERROR);
    Task_ID = find(min_ERROR==min(min_ERROR));
    ERROR=data{Task_ID,4};
    [xm,ym] = find(ERROR==total_min);
clear data

    step = length(A)/size(parameter_a,1);
    A_task = A(step*(Task_ID-1)+1:step*Task_ID);
    f2=figure;
        [Bmesh,Amesh] = meshgrid(B,A_task);
        meshc(Amesh,Bmesh,ERROR)
        set(gcf,'Renderer','ZBuffer')
        hold on

        % Minimal Value in 3D graph
        plot3([parameter_a(Task_ID,1) parameter_a(Task_ID,1)],...
            [parameter_b(Task_ID,1) parameter_b(Task_ID,1)],...
            [min(ERROR(:)) max(ERROR(:))],'r-*','LineWidth',2)
        hold off
        xlabel('parameter a','FontWeight','bold');
        ylabel('parameter b','FontWeight','bold');
        zlabel('error','FontWeight','bold')
        print(f2,'-dpng','pic/error_surface32.png');

figure(f1)
    C  = my_function(y0,parameter_a(Task_ID,1),parameter_b(Task_ID,1),x);
    hold on
    plot(x,C,'r','LineWidth',2)
    legend('Real data','New Data')
    hold off
    print(f1,'-dpng','pic/data.png');


disp('done')
```

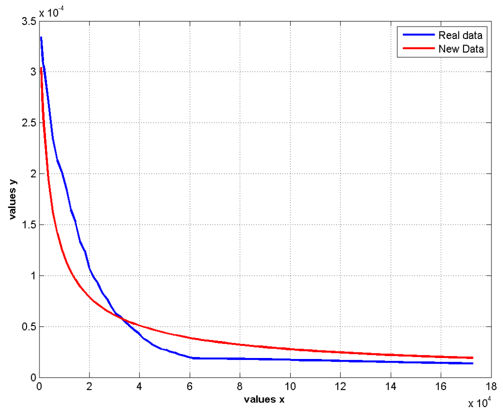


Figure 2: Real measured data and approximated data

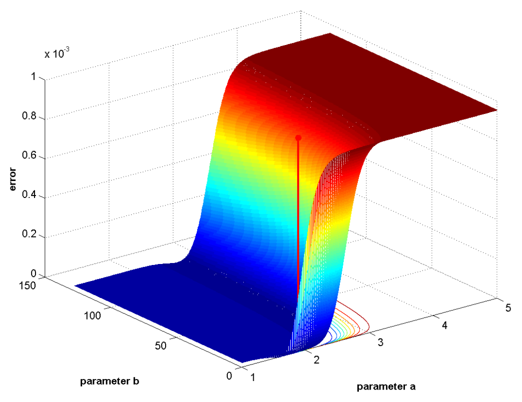


Figure 3: Error surface with minimal error

4 Monitoring of Cluster Calculations

4.1 Third Party Monitoring Tools

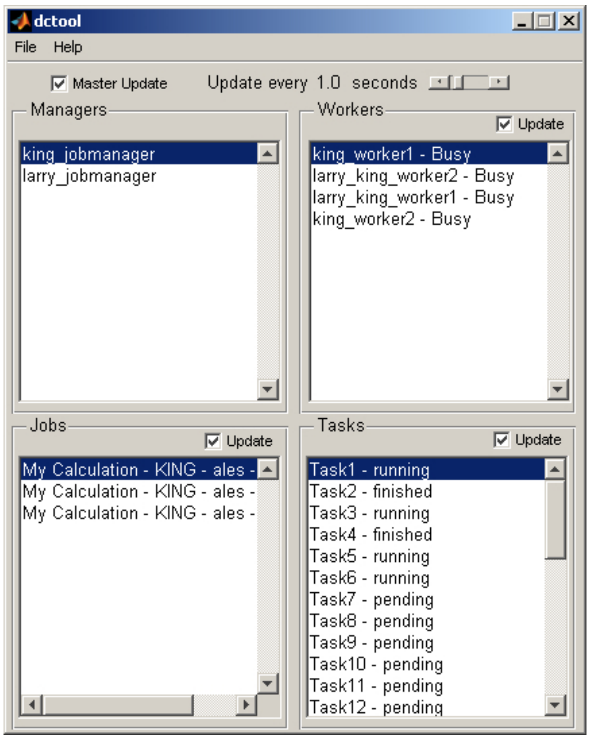
When the *job* is submitted to the *job manager* you can close MATLAB session and without problem receive current job status or results later. Command `jm = findResource('jobmanager');` is the most important one. After applying of `getjm` we receive structure array from which we obtain any information that is in the *job manager*. For comfort *job* and *tasks* monitoring it is suitable to use shared files in MATLAB Central File Exchange.[2] Especially we recommend the following functions:

dctool.m - Graphical interface for displaying the status and managing of a distributed computing network (Fig. 4).[3] This GUI tool allows complete real-time monitoring of *jobs* and *tasks*. It enables to submit or destroy specific *job*, create *JobReport* (Fig. 5) that shows time consumption of all *tasks* in the *job* per *worker*.

listJobs.m - Displays a summary of *jobs* in the *job manager*.[4] This command line tool has one input - handle of *job manager* for which it shows the list of *jobs* (Fig. 6).

listTasks.m - Displays the summary of *tasks* belonging to a *job*.[5] This command line tool has one input - handle of *job* for which it shows the list of *tasks* (Fig. 7).

There is only one strong limitation of listed tools that is based on the network speed, number of *workers*, *jobs* and *tasks* in the *job manager*. The increasing number of those elements increase the response time of tools as well. This limitation also occurre when user explore those elements directly via exploring of structure array `jm`.



```
king_jobmanager
larry_jobmanager
```

```
king_worker1 - Busy
larry_king_worker2 - Busy
larry_king_worker1 - Busy
king_worker2 - Busy
```

```
My Calculation - KING - ales -
My Calculation - KING - ales -
My Calculation - KING - ales -
```

```
Task1 - running
Task2 - finished
Task3 - running
Task4 - finished
Task5 - running
Task6 - running
Task7 - pending
Task8 - pending
Task9 - pending
Task10 - pending
Task11 - pending
Task12 - pending
```

Figure 4: GUI of the dctool program

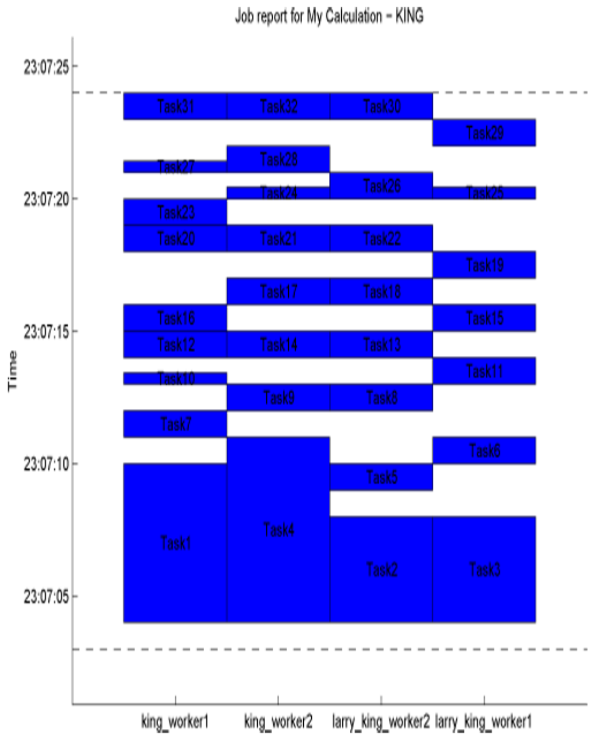


Figure 5: The Job Report figure created by the dctool

[2]http://www.mathworks.com/matlabcentral/fileexchange/
[3]http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=8056&objectType=file
[4]http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=9722&objectType=file
[5]http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=9722&objectType=file

```
Summary of jobs currently in job manager 'king_jobmanager':

                                                                            Tasks    Tasks     Tasks
Job ID              Job name  Username  Job state  Entered state  pending  running  finished  Errors
------  --------------------  --------  ---------  -------------  -------  -------  --------  ------
    37  My Calculation - KING      ales    running   Oct 01 23:10       22        4         6       0
    34  My Calculation - KING      ales   finished   Oct 01 23:06        0        0         8       0
    35  My Calculation - KING      ales   finished   Oct 01 23:07        0        0        32       0
                                                                  -------  -------  --------
Totals                                                                 22        4        46

Job manager state:       running
Number of busy workers:        4
Number of idle workers:        0
Total number of workers:       4
```

Figure 6: The Job Report figure created by the dctool

```
Summary of tasks belonging to job 'My Calculation - KING' (job ID 37, username ales):

Task ID  Task state  Entered state  Worker  Function name  Error identifier
-------  ----------  -------------  ------  -------------  ----------------
      1    finished   Oct 01 23:11    king        my_Task
      2    finished   Oct 01 23:11   Larry        my_Task
      3    finished   Oct 01 23:11    king        my_Task
      4    finished   Oct 01 23:11   Larry        my_Task
      5    finished   Oct 01 23:11   Larry        my_Task
      6    finished   Oct 01 23:11   Larry        my_Task
      7     running   Oct 01 23:11    king        my_Task  (task still running)
      8     running   Oct 01 23:11    king        my_Task  (task still running)
      9     running   Oct 01 23:11   Larry        my_Task  (task still running)
     10     running   Oct 01 23:11   Larry        my_Task  (task still running)
     11     pending   Oct 01 23:10                my_Task  (task not started yet)
     12     pending   Oct 01 23:10                my_Task  (task not started yet)
     13     pending   Oct 01 23:10                my_Task  (task not started yet)
    ...
```

Figure 7: The Job Report figure created by the dctool

4.2 Results Monitoring

Another way how to easily monitor the current status of *job manager* is in the use of the monitoring tool based on the previously mentioned third party tools that would be created for the MATLAB Web server. There is also opportunity to include in such interactive web tools monitoring, starting and stoping of *workers*, in the computation cluster.

The next way that we have realized is tightly connected with already finished *tasks* and of course probably will not be very suitable for smaller calculation. The main idea is in the storing of *tasks* results in a specific place. This place can be realized by the FTP server or the mailbox. We have possibility to send mails, download files from web pages, evaluate ftp commands directly from MATLAB environment. We have used this feature for storing all results from *tasks*. The following code presents how to easily save results and to upload them using a specific ftp server. The given code should be implemented in the code for the *task* function. As it is obvious this way of results monitoring can give us only information about just successfully finished *tasks* and their final times.

```
%% Final Operations
   save(['Task_',nameCode,'.mat']) zip(['Task_',nameCode,'.zip'],...
      {'Task_*.mat','*.tex','*.png'})
% Conection to to the localhost ftp server as anonymous user
  ftp_connection = ftp('localhost','anonymous','ales.pavelka@gmail.com');
  cd(ftp_connection,'upload');
  binary(ftp_connection);
  mput(ftp_connection,'*.zip');
  close(ftp_connection);
% Clean up
delete(['Task_',nameCode,'.zip'],'Task_*.mat','tab*.tex','*.png')
```

5 Application of Distributed Computing for Signal Prediction

5.1 Methods and Algorithms

Our prediction system is able to find out the most suitable parameters and model architecture for the following models: simple approximation model, autoregressive model (full, limited), neural network (linear, feed-forward) and model used in the UDP[6] for gas prediction.

Simple approximation model is based on the polynomial fit of values of gas consumption against values of daily temperature. The resulting model is the polynomial of the n-th order.

The UDP model is a real model that is practically used for gas consumption prediction. This model is based on the linear approximation of the specific input data set.

The autoregressive model (AR) has output signal $\underline{y}$ and unmeasured inputs called disturbance or noise $\underline{e}$ and n_a elements of parameter vector $\underline{a}$ in the form defined by relation

$$y(n) + a_1 y(n-1) + \ \dots\ + a_{n_a} y(n - n_a) = e(n) \tag{1}$$

In our calculations we use full autoregressive models alike as limited (cut, sub-set) models that come out of full models but have limited number of parameters. This limitation is performed by combination of autocorrelation, singular value decomposition and QRcp factorization.

The main part of our work is devoted to neural networks. The linear neural network is formed by a one layer structure with its linear transfer function having the unknown slope. Problem of finding minimal error is usually solved by the Widrow-Hoff algorithm. And thereby this neural networks is by its own concept very close to the autoregressive models.

A feed-forward neural network we used with the two layer architecture. The first transfer function was formed by a hyperbolic tangent sigmoid function and the second one by an unlimited linear function with the unknown slope. The Levenberg-Marquardt backpropagation method has been used as a basic learning algorithm for this network type.

5.2 Results

Practical calculation in cluster of eights computers has been done to solve the prediction problem and suitable parameters have been searched for six prediction models. Original plan was to calculate six models for 5 time zones (10.11.2003 – 10.04.2004, 10.12.2003 – 10.04.2004, 10.01.2004 – 10.04.2004, 10.02.2004 – 10.04.2004 and 10.03.2004 – 10.04.2004) that determine 5 separate *jobs*. In one *job* we created optimistically 8232 *tasks* searching suitable model for values of gas consumption (1–14), values of daily temperature (1–12), number of prediction outputs (1–7), and lengths of prediction (1–7). Number of prediction outputs means that in one prediction step the model predict possibly 7 values ahead e.i. the model has 7 outputs. The length of the prediction determine how many values will be totaly predicted by all types of the given model.

After several hours of calculation in our cluster the whole system hang. This interruption has been probably done by the overloading of *job manager*, but the real cause is unknown. Obtained results (2962 files) form just a small piece of previously planed calculation for this article. All results have been process and the best model for every model type has been selected. The main key for model selection included the minimal values of MSE criteria, according to this models presented in Tab. 1, Tab. 2 and in Fig. 8-13. Parameters of the best models are organized in Tab. 1. It is obvious that from all calculated models the best ones predict one single value to the future only.

Number of hidden neurones for feed-forward neural network has been set automatically to the value 1, information about day is one of input parameters, model limitation for autoregressive model has been set automatically to the value 9.

[6] Ústřední plynárenský dispečink - Bilanční centrum (http://www.bilancnicentrum.cz/)

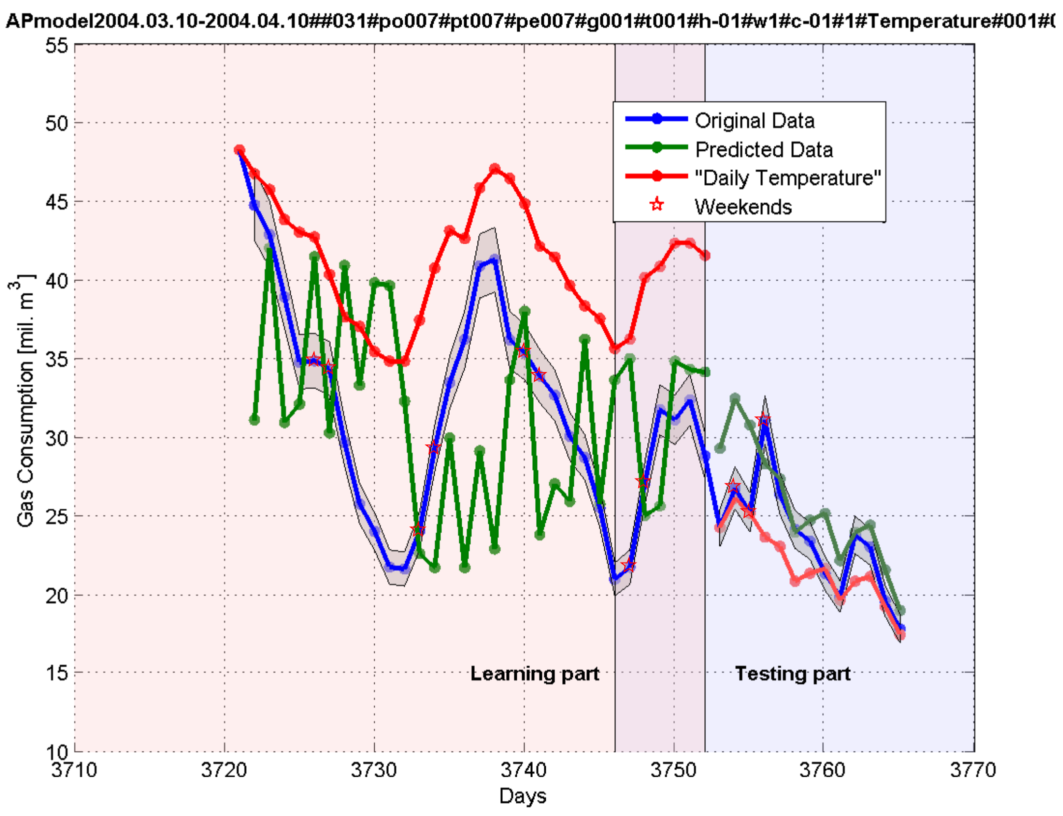


Figure 8: Prediction results for polynomial model.

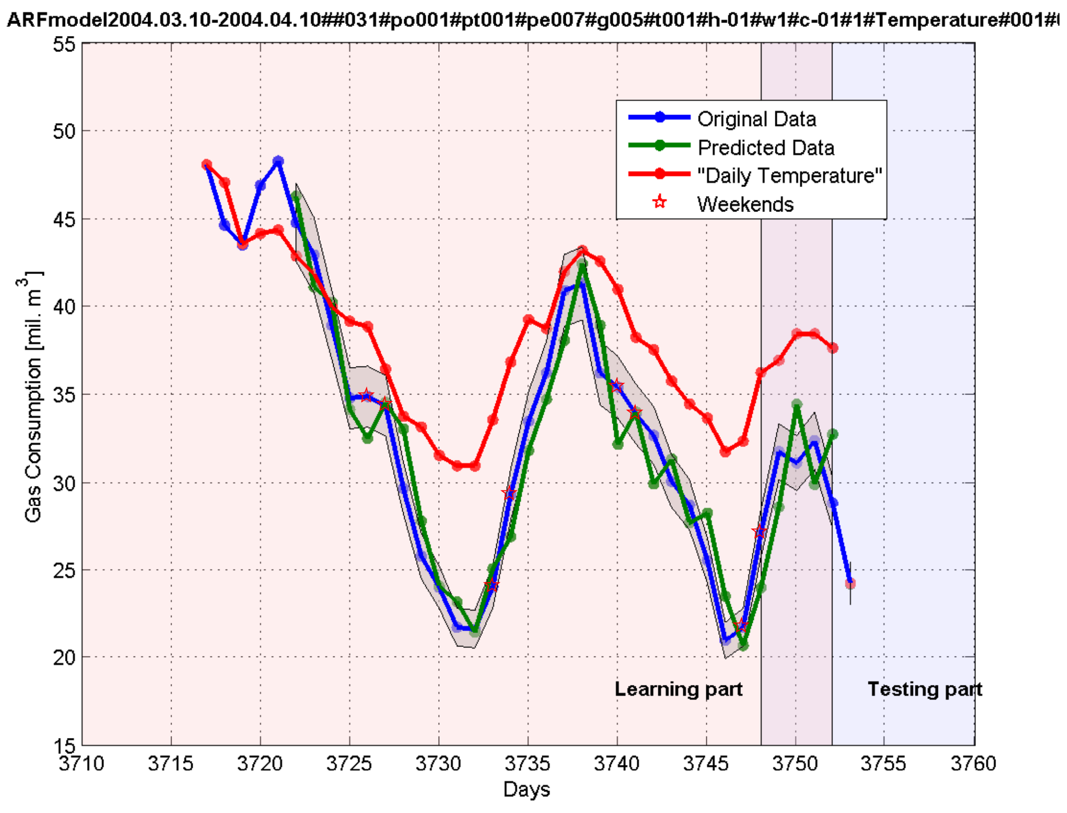


Figure 9: Prediction results for autoregressive model - full.

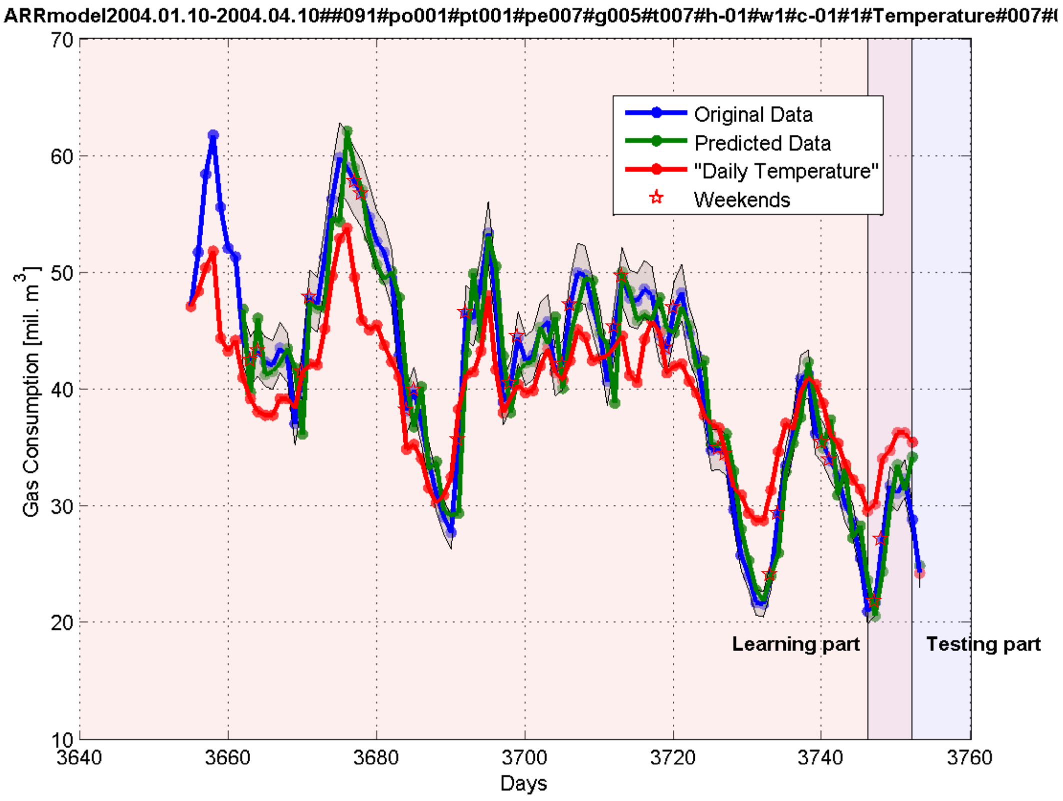


Figure 10: Prediction results for autoregressive model - short.

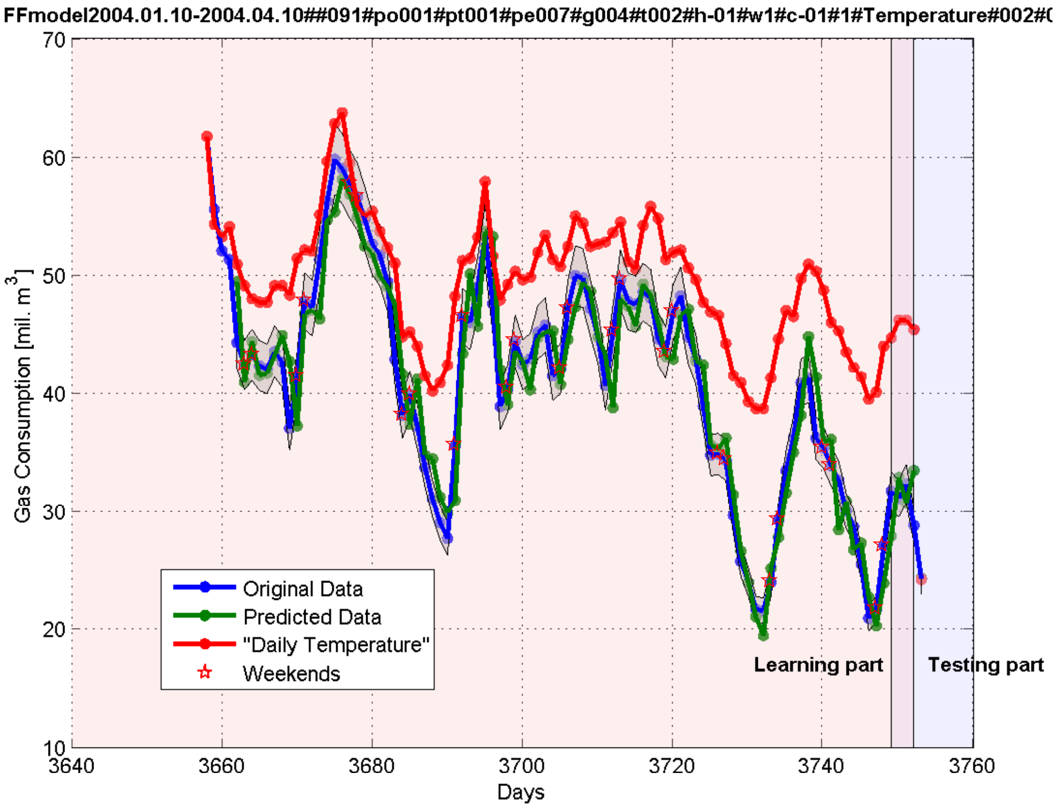


Figure 11: Prediction results for feed-forward neural network.

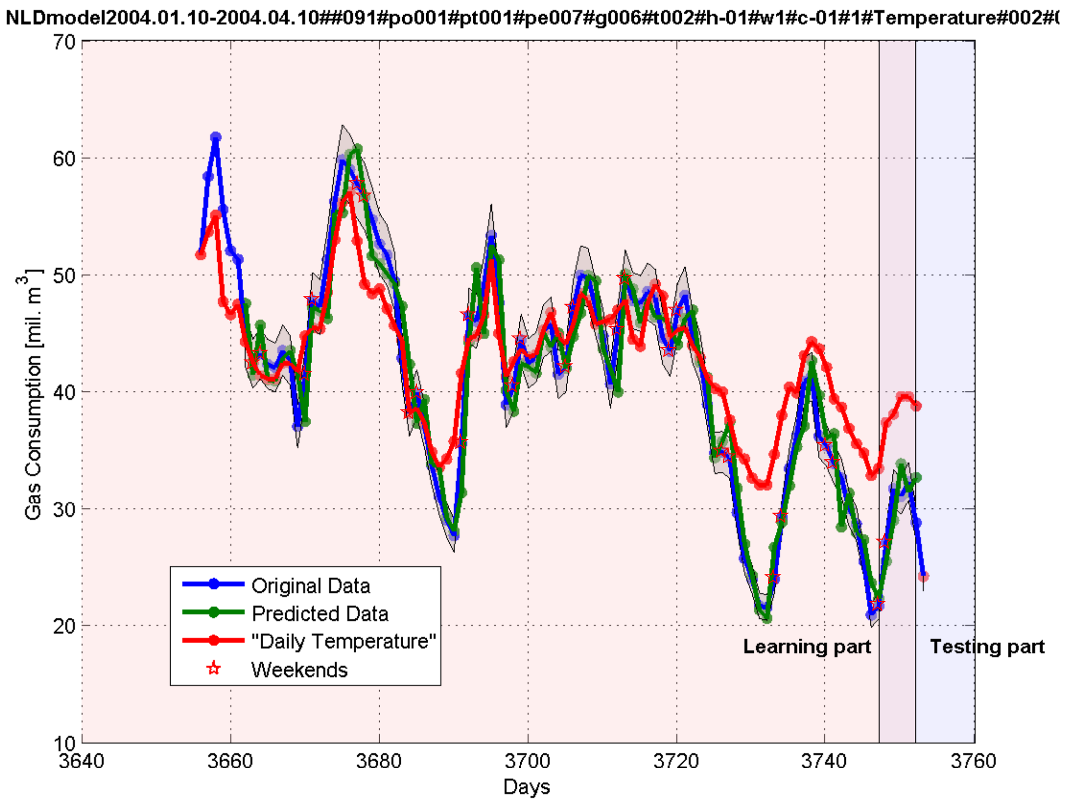


Figure 12: Prediction results for linear neural network.

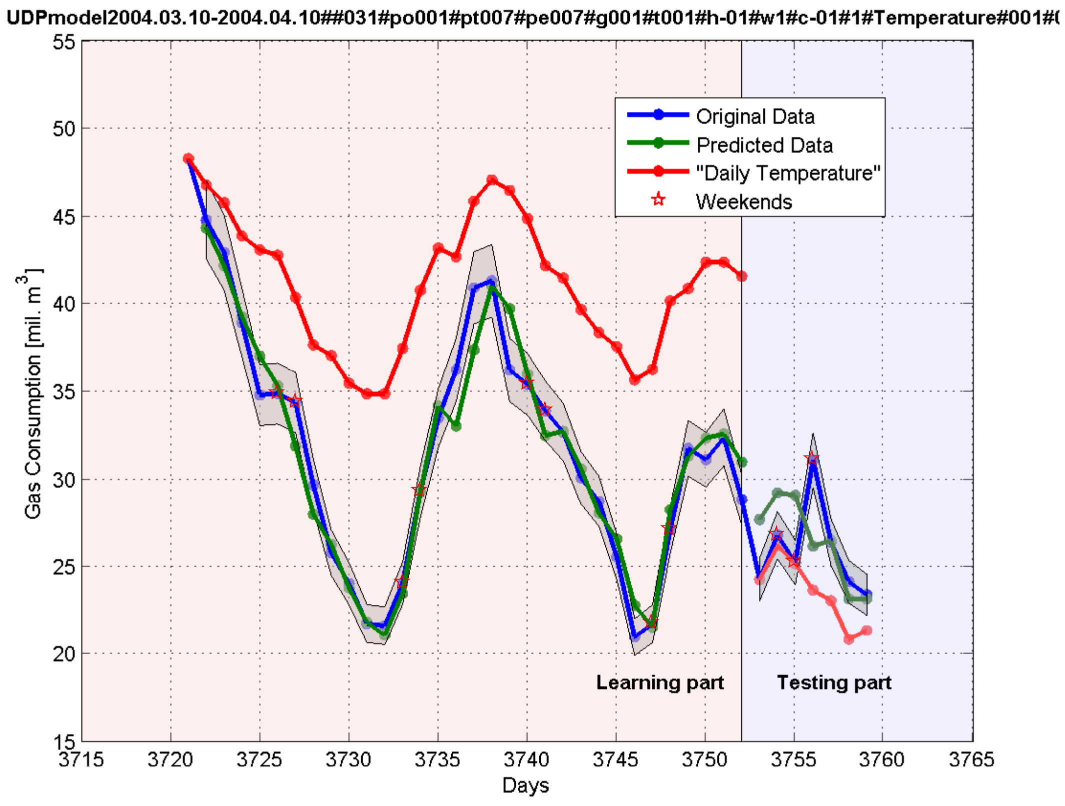


Figure 13: Prediction results for UDP model.

Table 1: Parameters of selected models

model	days	number of days	prediction output	prediction time	gas	temperature
polynomial	2004.03.10 – 2004.04.10	31	7	7	1	1
AR full	2004.03.10 – 2004.04.10	31	1	1	5	1
AR short	2004.01.10 – 2004.04.10	91	1	1	5	7
feed-forward NN	2004.01.10 – 2004.04.10	91	1	1	4	2
linear NN	2004.01.10 – 2004.04.10	91	1	1	6	2
UDP model	2004.03.10 – 2004.04.10	31	1	7	1	1

Table 2: Prediction error of selected models

model	mean	std	min	max	SSE	MSE
polynomial	-2.029	2.5	-5.7	2.749	125.262	9.636
AR full	0.051	0	0.1	0.051	0.003	0.003
AR short	-0.640	0	-0.6	-0.640	0.409	0.409
feed-forward NN	-0.120	0	-0.1	-0.120	0.014	0.014
linear NN	-0.049	0	-0.1	-0.049	0.002	0.002
UDP model	-0.522	3.0	-3.9	4.913	57.500	8.214

6 Conclusion

Calculation of our example of investigation of 12 800 $\times$ 12 800 matrix takes on one computer approximately more than 4 hours on the contrary with approximately less than 30 minutes of calculation time in the cluster of 8 computers.

Installation, setup and managing of MATLAB cluster created by computers with DCE and clients with DCT is very intuitive and fast. Very likely as basic programming principles of distributed programming that is strongly supported by the first-quality documentation.

According to our experiences it is suitable to create *tasks* that have duration lower than 2-3 minutes. Creating small, faster *tasks* increases the number of *tasks* and consequently some time is lost in network communication among *job manager* and *workers* and of course between *job manager* and your *client* station.

The total number of *tasks* in one *job* should not be higher than 1024. The reasons why not to create more *tasks* in one *job* are similar as in the previous case.

References

[1] The MathWorks. *Distributed Computing Toolbox, Users Guide.* The MathWorks, Inc., Natick, MA, version 2 edition, March 2006.

[2] The MathWorks. *MATLAB Distributed Computing Engine, System Administrators Guide.* The MathWorks, Inc., Natick, MA, version 2 edition, March 2006.

Ing. Aleš Pavelka, Prof. Aleš Procházka
Institute of Chemical Technology, Prague
Department of Computing and Control Engineering
Technická 1905, 166 28 Prague 6
Phone.: 00420-2-2435 4198, Fax: 00420-2-2435 5053
E-mail: ales.pavelka@gmail.com, A.Prochazka@ieee.org
WWW: http://dsp.vscht.cz