RUNNING THE NeuroWeb APPLICATION ON THE VIRTUAL EZ-GRID PLATFORM

INTERNAL TECHNICAL REPORT - THE VIRTUAL EZ GRID PROJECT
Mohamed Ben Belgacem, Cédric Bilat, Marko Niinimaki and Nabil Abdennadher
hepia, HE-Arc, January 2011

This report describes the gridification of the NeuroWeb application and its deployment of the XtremWEB-CH (XWCH) desktop computing platform. This work was carried out in the framework of the AAA/Switch project: Virtual EZ Grid.

The document is organized as follows: Section 1 details the functionalities of the NeuroWeb application and the context in which it is used. Section 2 describes its gridification. The deployment and the experimental results are presented in section 3. Finally, Section 4 presents some perspectives of this work.

1. Background

The *Neuroweb* application proposes to build neuronal maps of brain activities using the input from non-invasive measurements [1]. It is based on "extracting" the activity of the different regions of neurons from captors attached on the scalp of the patient. Measurements of the internal cerebral activity around the scalp are obtained, for example, from a Magnetoencephalography (MEG) (figure 1).

Neuronal activities, called generators, are electromagnetic fields captured by the MEG captors. These fields are extremely noisy and disturbed by the cell tissue. In essence, we have a lot of generators (~60.000) but very few captors (~200). The difficulty, and hence the computational requirement effort, is to construct a robust and efficient estimator for a neuronal map [2]. NeuroWeb could be used to identify sources of brain "abnormal" activities such as epileptic crisis, Parkinson, Alzheimer, etc. The NeuroWeb application produces a dynamic neural map, DNM, in which each row represents the electromagnetic activity of one region of neurons (generator). With 60.000 regions and 1500 measures (assuming that the duration of the experiment is 1.5 second and one measure is taken every 1 millisecond), the size of DNM is greater than 240 MB.

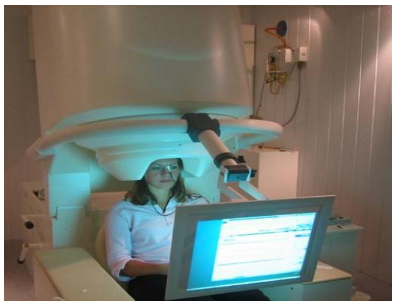


Figure 1: Magnetoencephalography (MEG)

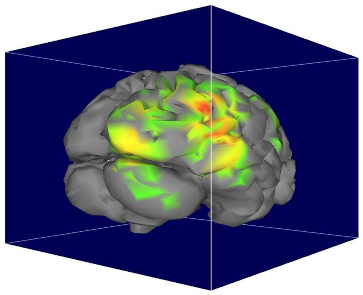


Figure 2: A dynamic neuronal map

The neuronal map extracted from brain measurements and calculated with the NeuroWeb grid software will be integrated into the BrainStorm (http://neuroimage.usc.edu/forums/, 2.600 users) and the NeoBrain software[1] to visualize the spatial and temporal brain activity in 3D stereo world (Figure 2). NeoBrain allows a smooth and interactive visualization, and enables a fine, precise and convenient analysis of this important quantity of spatial and temporal images, previously grid calculated. For example, the neurons responsible for some diseases (like Parkinson's and alzheimer... etc.) are represented by the red color in the figure 2.

Algorithm:

NeuroWeb uses an iterative and nondeterministic algorithm, aiming to progressively construct a matrix data which represents brain activities [3]. It starts from an initial matrix called DNM0 and data measurements from scanners MEG (magnetoencephalography) and MRI (Magnetic resonance imaging). At each step i, a new matrix DNMi is constructed based upon DNMi-1. The iterations (steps) stop when the DNMi matrices can no longer be improved, i.e., when convergence is reached (Figure 3). This matrix calculation is a heavy and CPU consuming task: one matrix of small brain (45 Mo) takes 60 minutes on a normal PC (Dell xps M1520 with Ubuntu 10.04 Desktop edition, 3Gb RAM, dual core CPU 2x2GHz), therefore a solution for paralellization is needed.

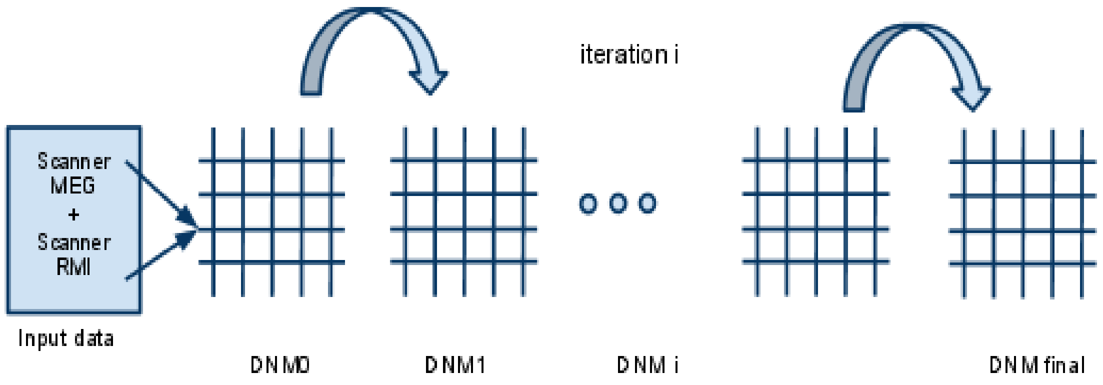


Fig3. NeuroWeb sequential algorithm

The basic solution consists in gridifying each step by splitting the DNM matrix into several blocks Bj . Each block is processed by a daemon process that executes an iterative asynchronous algorithm. The daemon process is associated to an XWCH persistent server (PS). During its execution, a PSj receives data from its neighbours (PSj-1 and PSj+1) and integrates them to calculate a new "improved" version of the block Bj. In one given block Bk, columns Ck-1 and Ck+1 are required to process column Ck. This means that PSj and PSj-1 need to exchange a common column, and so do PSj and PSj+1. It’s worth reminding here that the generation of a block Bi,j (block generated at stage i by the PSj) can take place even if no input data are received from the neighbours of PSj . Indeed, PSj is an iterative asynchronous algorithm which continues improving its block Bj even if it does not receive any data from its neighbours.

[1]NeoBrain is a software developed by the University of Applied Sciences, Western Switzerland (HES-SO) with the collaboration of the Swiss Federal Institute of technologies (EPFL), and the participation of Hopital Pitié Salepétrière of Paris.

On the other hand, a central persistent server (CPS) is deployed in order to gather blocks from all PS's and decide if convergence is reached or not. If the convergence has not yet been reached, the CPS launches a new step.

2. Gridification

2.1 Dataflow graph

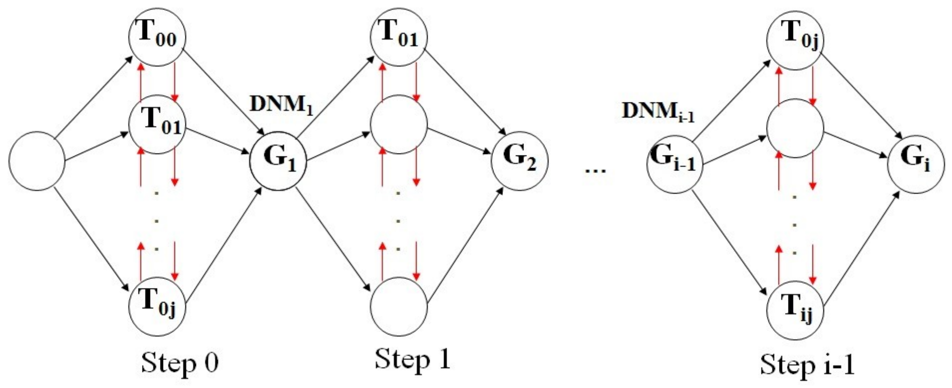


Fig4. DataFlow graph

The dataflow graph of the NeuroWeb application is represented by Figure 4. As shown, there are two types of tasks: Tij which feed the PSs with data and Gi which are responsible to transfer data to the CPS. In order to avoid unnecessary data transfer, tasks Ti0, Ti1,...,Tij should be executed on the machine where the corresponding PSi is running. Moreover, a PS should receive its data from its neighbours and/or from the CPS, in a well-defined order, and feed them with necessary data after processing. In what follows, we explain how the does the NeuroWeb application manages the PS's and the CPS and how data are transferred among these modules.

2.2 persistent servers

The first step of running NeuroWeb is to create the PS's on remote workers. A PS is a Java daemon that provides mainly the following local services:

- Init(): the PS is informed with input files: initial matrix bloc, input data, PS's number (Fig5,(1)).
- TransferData(): Informs the PS about incoming data from its neighbours and/or CPS (Fig5,(2),).
- Finalize(): ends the calculation and kills the PS (Fig5,(3)).

Figure 5 shows how a PS is created on a remote worker.

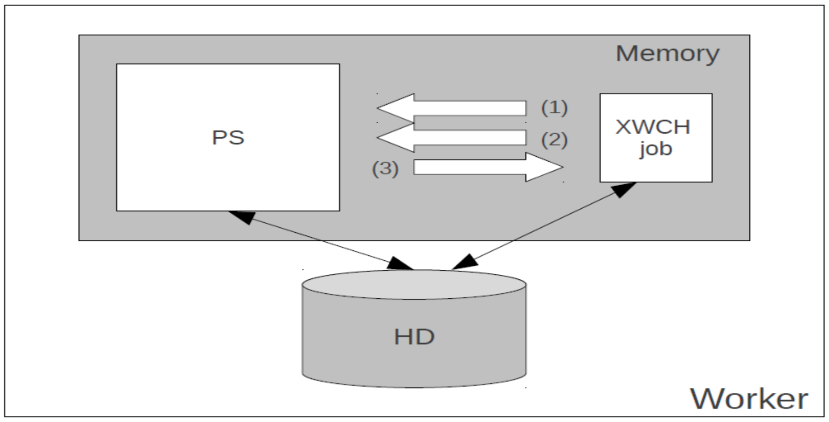


Fig5. PS creation

Starting from the client machine, a Java based client program, written using the XWCH Java API [4], sends an XWCH task that creates the CPS and PSs, then waits it to finish. The input data for this task is the initial DNM0 matrix, the scanners measurement data and the number **k** of PS to create. Once running on a worker, the CPS splits the initial matrix into **k** blocks {Bj}. Each Bj defines the input data for a new XWCH task that is sent by CPS to create a new PS on an remote worker. The CPS remains running until the end of the calculation. As illustrated in Figure 5, when an XWCH task arrives on a worker, it creates in the background the PS which remains running even if the task finishes. Consequently, future XWCH tasks will invoke the PS without needing to read/write from the disk.
When the convergence is attained, the client program kills the CPS and all PS's and retrieves the final matrix file from warehouses on the client host (4).

2.3 Client program

The client program aims mainly to handle the well-defined order constraint of tasks. It consists of one Producer and k Consumers, which share the same list of queues. Each queue is dedicated to a PS and managed by a Consumer. These components are running on the CPS side. The Producer and Consumers are simply Java daemon processes. The data to transfer between two PCs is represented by a Messages list: in NeuroWeb context, all data exchange are done through Messages, i.e. Java Objects that contain: the sender PS, the destination PS and embedded data to transfer.

To start, an ordered queue list of message are constructed on the CPS side, by assigning a queue to each created PS. Messages, when arriving, are dispatched based upon the “destination” parameter in the message object. The producer dispatches messages into the appropriate queue according to their arrival date (FIFO behaviour). At the same time, the k consumers are consuming queues in the following way:

- Retrieve a message list L from the queue.
- Send L to the appropriate PS through an XWCH task.

The output file of a task represents the data to transfer to neighbours (PSs) and/or CPS; i.e. a list of messages which will be dispatched by the producer on the queue list. This asynchronous system is well suited to handle the well-defined order constraint of the incoming messages. Indeed, in order to respect this constraint, a list of Messages can't be polled from the queue unless its previous sent task has already been finished.

When convergence is reached, a special Message Object is sent to each PS to kill it.

Algorithm.

This tasks scheduling system is illustrated in Figure 6:

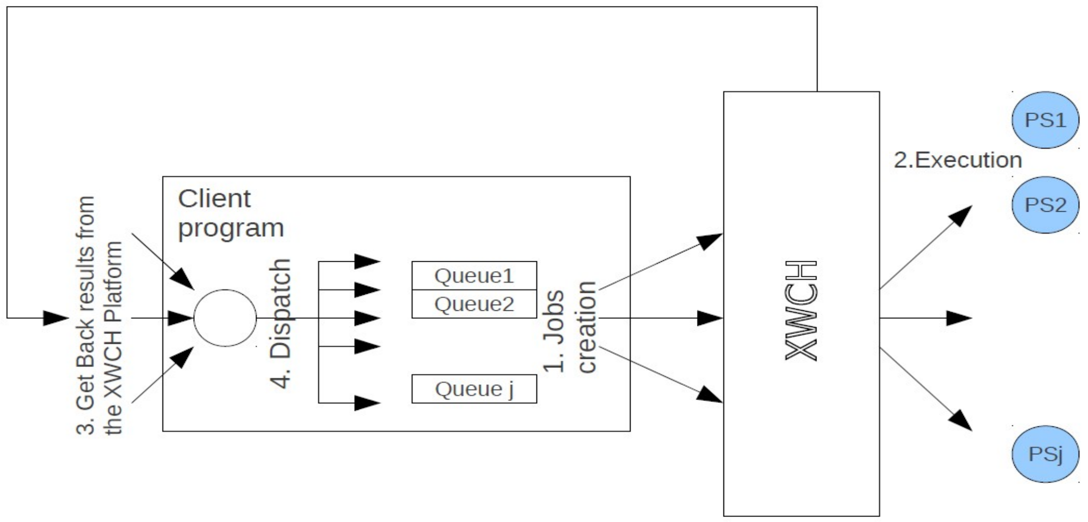


Fig 6. NeuroWeb execution lifecycle

Before starting the calculation, the CPS splits the initial matrix into j blocks, creates corresponding Messages list for each PS and inserts them on the queue list. The algorithm of the execution is described as follow:

In the above algorithm, each consumer retrieves only one output file per task, containing a list of Messages. For sake of optimization, it is not recommended to retrieve the data block within the XWCH task output file. “multi output files” feature was implemented for this purpose: an XWCH job can create more than one output file. The basic flexibility that makes this feature attractive for the NeuroWeb application is to separate data block from the Message object. As a result, data block remains on warehouse. This date is referenced by a pointer located in the Message object. In other words, for a PSi, a job creates three output files: “fout”, “fout0” and “fout1”. The output file “fout0” and “fout1” should be send to the PSi-1 and PSi+1 while “fout” contains Messages and only pointers to “fout0” and “fout1”, that will be retrieved by the client program.

3. Deployment and experiments

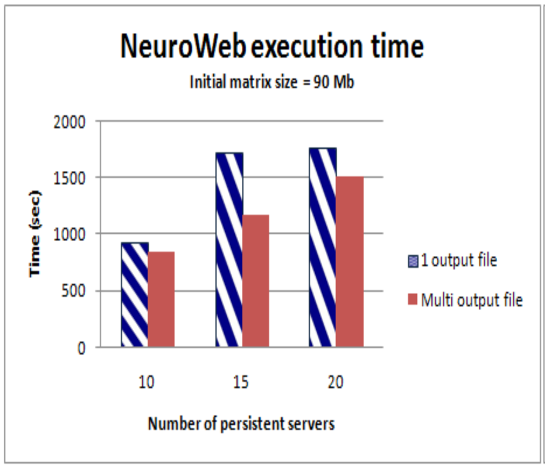
measurements of a small brain treatement were carried out on an the XWCH2 platform composed of 4 warehouses and about 500 connected workers. We have only used Linux workers, (2 cores CPU and 3 Gb RAM).

JVMs	Exec Time
5	00:26:17.25
6	00:22:11.50
7	00:23:13.38
8	00:17:38.49
9	00:16:47.51
10	00:18:10.79
11	00:22:10.40
12	00:21:29.16
13	00:23:09.37
14	00:27:04.83
15	00:20:31.63
20	00:17:18.69
25	00:12:31.18
30	00:12:25.38
35	00:18:37.46

table 1 : NeuroWeb execution time (samll brain, convergence parameter=1e-3)

Table 1 illustrates the impact of the number of workers used for the calculation. We can see that with an "epsilon score" (convergence parameter) equal to 1.e-3 and an initial matrix size equal to 43 Mb, the best worker number for minimal execution time is about 30 workers. Comparing to local execution on a normal pc (3Gb of memory, Intel dual core: 2x2GHz) that took 59 minutes, the use of XWCH speed up the execution time by a factor of 5. It’s worth reminding here that the main benefice that makes XWCH attractive for NeuroWeb application is the possibility to use more workers when needed: in case of a big matrix or simultaneous treatment of a set of brains.

Figures 7 illustrates the difference between two execution time: the single and multi output file. We think that the difference between the two execution time is not significant due to the fact that the DNA matrix corresponds to a small brain and to the heterogeneity of the workers.



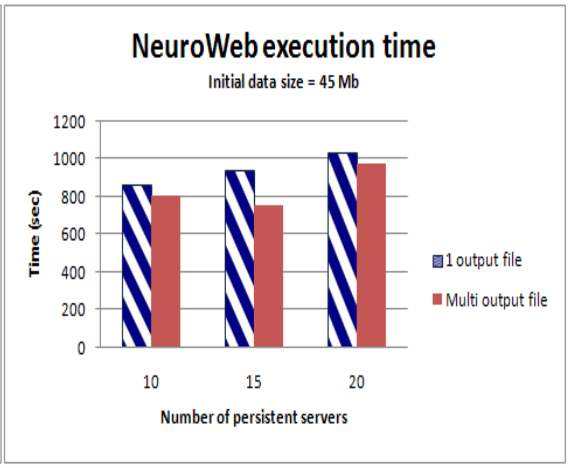


Figure 7: NeuroWeb execution time

It is worth noticing here that the PSs are not visible in the job statistic graphs (http://www.xtremwebch.net/stats/r2.html) of the platform since they are not XWCH jobs (they are created by XWCH jobs).

4. Conclusion and perspectives

In this report, we have presented the gridification of the NeuroWeb application on the Virtual EZ Grid platform. This gridication is based on the XWCH Java API. Moreover, we have tried to optimize the communication and data transfer for better performance. Practical experiments show that the most efficient number of workers for a small brain calculation is about 30 workers. Let us remind that several simulation for the same data should be carried by XWCH tasks to obtain the best results.
Works is currently under way to integrate the gridified version of the Neuroweb into the Neuroweb prototype[2]. This will enable end users (medical staff) to start using the system with a real data.

We will be further enhancing our experiments by gridifying the NeuroWeb application on a cluster. We also intend to make a performance comparison of running NeuroWeb both in volunteer computing and cluster environments.

References

[1] N. Abdennadher: Using the Volunteer Computing platform XtremWeb-CH: Lessons and perspectives, ACSE'09, 2009
[2] N. Abdennadher, C. Evéquoz and C. Billat: Gridifying phylogeny and medical applications on the volunteer computing platform XtremWeb-CH, Stud Health Technol Inform, 2008

[2]NeoBrain is a software developed by the University of Applied Sciences, Western Switzerland (HES-SO) with the collaboration of the Swiss Federal Institute of technologies (EPFL), and the participation of Hopital Pitié Salepétrière of Paris.

[3] M. Niinimaki, M. Ben Belgacem, N. Abdennadher : Programming distributed medical applications with XWCH2. *Stud Health Technol Inform*, V(159): 100-110, 2010.
[4] XWCH Java doc. http://www.xtremwebch.net/javaapi/javadoc/

Appendix

Package architecture:

The NeuroWeb package has the following tree architecture:

```
 +____ data.zip
 |      |___ libraries.zip
 |      |      |___ junit-4.5.jar: used to test assertions
 |      |      |___ ALL_HEARC_O.jar: contains all functional NeuroWeb services
 |      |      |___ invokeworker.jar: used to communicate with a PS (RMI and PIPE)
 |      |
 |        |___ CreateJVMWorker_LINUX.zip
 |      |      |__ runw.sh : creates a PS
 |      |
 |      |___ InvokeJVMWorker_LINUX.zip
 |             |__ invoke.sh : used to communicate with a PS
 |
 +___ binarymaster.zip
 |      |___ runmaster.bat : creates the CPS
 |      |___ StartD12Launcher.jar : used to create the CPS
 |
 +___ bin/
  |      |___ ALL_HEARC_O.jar
  |      |___ D12LanceurXWCHs.jar  : the main package to start the calculation
  |      |___ junit-4.7.jar
  |      |___ XWCHClientAPI_V1.0.jar : XWCH API
  |      |___ configure.properties: runtime configuration
  |      |___ restor.log: off line result-retriever
  |
  +__ run_XWCH_NeuroWeb.sh: script to start the calculation from user's host.
```

The gridification of NeuroWeb is a collaboration between hepia and HES_ARC (www.**he-arc**.ch). HES_ARC provides a pre-compiled jar library called “ALL_HEARC_O.jar”, containing all NeuroWeb services, and hepia provides the main package “D12LanceurXWCHs.jar” that runs the NeuroWeb gridification over XWCH.

The Main class of the package “D12LanceurXWCHs.jar” is “D12LanceurXWCHs”. It inherits from the Interface “D12Lanceurs_A” (package “ALL_HEARC_O.jar”), which provides services like splitting initial matrix, directives to create PSs,... etc.
The main role of the D12LanceurXWCHs class is to :

- Create the CPS and all the PSs on remote workers.
- Hold the communication between all PSs.
- Detect convergence, retrieve final matrix file and end the calculation.

Running NeuroWeb

The “run_XWCH_NeuroWeb.sh” scripts starts the calculation. It calls the D12LanceurXWCHs class which serializes the the initial matrix and directives into a file “ser.obj”. Then, it creates a XWCH task **T0** with the following settings:

- taskname=”NeoBrain-XtremWebCH”
- module=binarymaster.zip
- input= “ser.obj”+ “bin” directory +data.zip
- cmdline=runmaster.bat ser.obj

When arriving on a worker, the sent task will start the NeuroWeb calculation:
1. Create the CPS and the scheduling system.
2. Create all PSs: for each PSj, a xwch tasks is created with following setting:

- taskname= “persistent server j”
- module=”CreateJVMWorker_LINUX.zip”
- input= {block Bj of matrix}+ libraries.zip
- cmdline= runw.sh ...

3. Hold the PSs communication: in order to communicate two PSs, a xwch task is created by the scheduling system with the following setting:

- taskname= “[Wi -->Wj]”
- module=InvokeJVMWorker_LINUX.zip
- input= {pointers to files}+{data from CPS if exists}
- cmdline= invoke.sh ...

4. Detects convergence and ends calculation by sending “kill” tasks.

On the client’s host, the client program retrieves the output file of the task **T0 (**final matrix file) when it ends.

UML class diagram (D12LanceurXWCH.jar)

The general UML class Diagram consists mainly of :

- *D12Lanceurs_A*: Java interface which declares all usable services: splitting matrix, preparing PSs data, ...
- D12Lanceurs_XWCH: implements some of *D12Lanceurs_A* services
- *XWCHLauncher* class: the main class that starts the calculation. it creates the CPS (on the same machine) and all the PSs, then, the *MetaScheduler* and the *AgentExecutors* objects.
- *MetaScheduler* class: implements the Producer. It maintains a list of AgentScheduler objects. When notified, the *MetaScheduler* dispatches messages in the appropriate queue list.
- *AgentSheduler* class: implements a consumer. A consumer is a pooling java thread that has a queue list. The AgentScheduler:
    1. Polls a list of message from the queue
    2. Submits an XWCH task and waits for its execution
    3. Processes the out file and generates a list of messages.
    4. Notifies the *MetaScheduler* about incoming list of messages.
    5. if (not convergence) go to 1) else end
- *MessagesLight* class: it represents data to transfer between two PSs. it has mainly the following attributes:
    - sender: int
    - receiver: int
    - block: pointer to file name+md5

The script “*run_XWCH_NeuroWeb.sh*”

This script runs on the client host and starts the NeuroWeb calculation.

```
 #!/bin/sh
 # gridificationType :   XWCH || LOCAL
 # nbTask in [2,N]
 # epsilonScore>0
# nbRepetition>=1
# nbSommet>=9
# nbActivation=1500
# isGuiEnable : true or false
# -connexionJvmType: RMI or PIPE
      java -Xmx500m -Djava.library.path=./bin -classpath .:./bin/*
   com.bilat.neoBrain.moo.carte.grid.serveurPersistant.concret.neobrain12.use.UseNeoBrain12s
   -gridificationType=XWCH -nbTaskMin=5 -nbTaskMax=5  -nbRepetition=1 -epsilonScore=1e-3
   -nbSommet=6000 -nbActivation=1500 -isGuiEnable=false -connexionJvmType=RMI
```