



User Manual

by Elmar.Krieger@yasara.org

Date of last revision: 2012/01/06

NOTE: Models@Home has been developed in 2000 at the CMBI/Radboud University Nijmegen, and has since been used for all computationally intensive tasks at YASARA.org, like force field parameter optimization or large scale homology modeling. Models@Home is however not part of the YASARA program and therefore not covered by YASARA support agreements. The source code is available so that you can fix any problems yourself, feedback is of course appreciated.

Models@Home

Distributed Computing in Molecular Modeling (and anywhere else)

With 40 human working hours per week as a well established upper limit, computers are typically idle at least three quarters of every day. To make optimum use of the available resources, one would like to reduce this unproductive time to a minimum. The idea to build a network of idle computers and let each of them work on a tiny sub-piece of a scientific challenge has led to the world's largest distributed computing cluster: Seti@Home, with currently more than three million participants analyzing radio signals on search for extra terrestrial intelligence (setiathome.ssl.berkeley.edu).

Models@Home follows this general approach, focuses however on the molecular modeling specific aspects of the problem. The two main differences are: 1) Modeling often requires an entire collection of programs to obtain a result. Adapting all these programs for parallel execution is either prohibitively time consuming or simply impossible (if the source code is not available). The Models@Home environment therefore allows to run any programs without modification - as long as they do not require direct user input. 2) Modeling applications are rarely as course-grained and insensitive to lost jobs as the Seti program. In most cases all jobs must be completed before a result is available, and many times the rate limiting step is not the number of computers, but the maximum time required to complete a single job. Models@Home therefore provides a more stringent control over job scheduling. In addition, the screen saver can be decoupled from the actual execution module for use in dedicated clusters. Models@Home is freely available for use in heterogeneous Linux/Windows environments at www.yasara.com/models

Table of Contents

Models@Home License.....	4
Overview.....	5
Supervisor Clients.....	5
Working Clients.....	6
Server.....	7
Installation.....	7
The steps to your own cluster.....	7
LINUX Server.....	7
LINUX Supervisor Client.....	8
LINUX Working Client.....	8
LINUX Working Client with screen saver.....	8
LINUX Working Client without screen saver, node in a dedicated cluster.....	9
WINDOWS Server.....	10
WINDOWS Supervisor Client.....	10
WINDOWS Working Client.....	10
WINDOWS Working Client with screen saver.....	10
Solving your problems in parallel with Models@Home.....	11
Installing your own programs on LINUX Working Clients.....	11
Installing your own programs on WINDOWS Working Clients.....	12
Parallelizing your inner loops on the Supervisor Client.....	13
The individual steps in Python.....	13
The individual steps in C.....	14
The central job queue cluster.job.....	14
The left side of cluster.job.....	15
The right side of cluster.job.....	15
Extending Models@Home to a large network.....	17
Automatic installation of programs at a later stage.....	17
Security issues.....	18
Useful hints to evade common pitfalls.....	18

Models@Home License

Models@Home consists of various modules, that are covered by different licenses, ranging from Public Domain to GPL and LGPL. A detailed list can be found in the file LICENSE.TXT. The Models@Home main programs "cluster" and "cluster.exe" are available under these license conditions:

- Models@Home is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY, without even the implied warranty of merchantability or fitness for a particular purpose. The copyright holders and/or other parties provide the data "AS IS", without warranty of any kind, either expressed or implied. The entire risk as to the quality and performance of the data is with you. Should any data prove defective, you assume the cost of all necessary servicing, repair or correction.
- Models@Home comes free of charge.
- Models@Home is not part of YASARA and not covered by YASARA updates and support agreements.
- Any description of work in which Models@Home was/is involved (scientific papers, websites etc.), should cite: E.Krieger and G.Vriend,"Models@Home - Distributed Computing in Molecular Modeling, Bioinformatics. 2002 18(2):315-8."
- The Models@Home, CMBI, WHAT IF and YASARA logos in the screen saver must not be removed or replaced. (But feel free to add your own logos.)
- The Models@Home source code "cluster.c" is available on a "tit for tat" basis with the following additional conditions:
 - You are allowed to modify/adapt the source code.
 - We help each other by swapping source code: For every change you make, please email the file back to Eliza@yasara.org with cluster.c in the email subject field.
 - If you add these changes to separate files and link them with cluster.c, you must also return these separate source code files.
 - By using Models@Home, you accept this license.

Overview

Models@Home has been implemented for use in heterogeneous Linux/Windows environments and consists of several functional units:

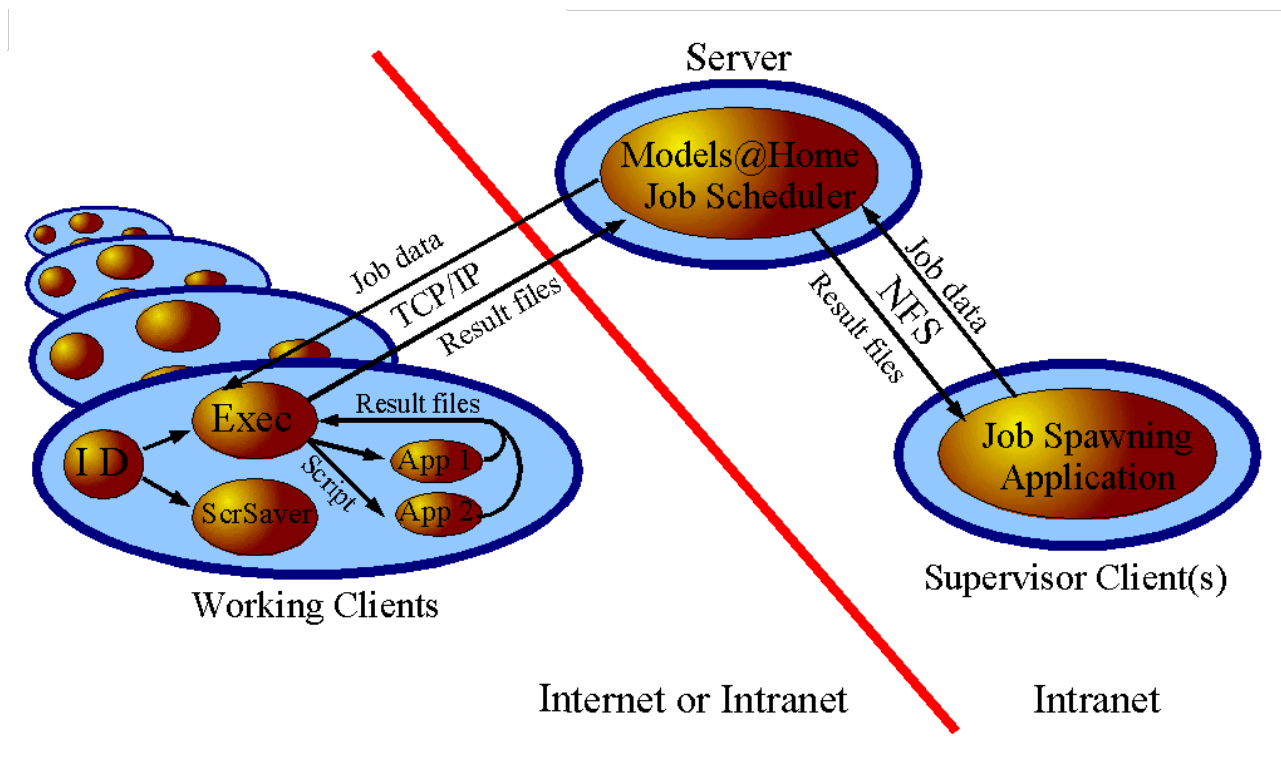


Fig.1: Models@Home data flow. Abbreviations: ID = Idle Detection module, Exec = Program execution module, ScrSaver = graphical screen saver module, App 1+2 = application 1+2, TCP/IP = Transmission Control Protocol / Internet Protocol, NFS = Network File System.

Supervisor Clients

These are typically applications that keep track of the work that has to be done, but do not do it themselves. Instead they cut it into pieces and submit the individual jobs to the Models@Home job scheduler (via an interface of C functions or a Python class). This simply works by locking the central job queue (a text file called "cluster.job") on the Server via NFS, adding the new jobs, and saving the file again. (In many cases, Supervisor Clients and Server will be running on the same computer, making NFS obsolete). Supervisor Clients thus form the part that has to be developed specifically for a certain application. This is usually done by modifying the inner loops of all those countless scripts filling up hard drives in bioinformatics labs.

Working Clients

As the name suggests, they do the actual work. Initially, only the Idle Detection (ID) module is active on these computers. The ID module is highly operating system specific and has been obtained by making minor modifications to existing open-source screen-savers. The Linux version is based on Jamie Zawinski's XScreenSaver (www.jwz.org/xscreensaver), the Windows equivalent on Bill Buckel's work (www.escape.ca/~bbuckels). The modified sources are available from www.yasara.org/models.

As soon as the computer is idle (i.e. no mouse movements or key-strokes occur for 15 minutes), the ID module launches a graphical screen-saver (ScrSaver) and the execution module (Exec). Both have been implemented in an operating-system independent way based on the SDL library, including SDL_net for the TCP/IP interface (www.libsdl.org).

The Exec module creates an empty working directory and contacts the Server. This message also contains the time stamps of files that should be kept up to date (these names are stored with other information, like the Server's IP address, in the local configuration file "cluster.cnf" (Linux) or "cluster-cnw" (Windows)).

If there are jobs waiting in the queue, the module receives a job description (the name of the application to run, command line parameters, scripts), the required file updates, and a number of files needed to complete the job, which are stored in the working directory. Exec then runs the requested application (App 1 or App 2 in fig.1) and waits until the job has been finished. Result files are sent back to the Server. The applications themselves must be installed on all the Working Clients if they are too large to be continuously transmitted via the net. If they are small, compact programs that fit into a single directory, they can be installed "on the fly" by simply including them in the list of transmitted files.

If a user on a Working Client terminates the screen-saver by pressing a mouse button, Exec kills the running application and notifies the Server that the job could not be completed. No attempt is made to put the application on hold and continue at a later time, as this would negatively affect the performance of both the Working Client (blocking memory) and the cluster (it is not known when the client will be able to finish the job). Long jobs must therefore checkpoint in reasonable time intervals, i.e. return a platform independent snapshot of their current state to the server.

Server

The Server is the link between Supervisor and Working Clients. It manages the job queue "cluster.job" and distributes jobs according to their priority. It also handles the transfer of job data files from a job-specific directory on the Supervisor Client (via NFS) to the working directory on the Working Client (via TCP/IP). Job results travel back in the reversed direction.

Additionally, the Server stores the latest file updates and transmits them to Working Clients to replace outdated versions. (In principle any file on the Working Client can belong to this update group, including the Models@Home software and the actual applications).

The Server also collects "still alive messages" from Working Clients. If a client does not send such a message for a given time period, it is assumed to have "disappeared without notice" (e.g. a power failure) and the job is retransmitted to another client.

Installation

Models@Home has been used at the CMBI in a heterogeneous Linux Red Hat 7.1-8.0, Suse 7.0-8.0, Mandrake 8.1- and Windows NT/2000 environment between 2000 and 2004. Since then, it has mostly been running on dedicated clusters, adding a few idle office PCs to the cluster when available.

The steps to your own cluster

- Choose two to three computers to start with: You need one server and two Working Clients to test it: One running Linux, one running Windows (unless you are only using one operating system).
- Download the Models@Home software from www.yasara.org/models. You will receive a ZIParchive cluster.zip (the same file for all operating systems).

LINUX Server

- Copy cluster.zip to any directory and unpack: **unzip cluster** . This will create a directory cluster and further subdirectories. (Unzip is part of virtually all Linux distributions.)
- Edit the central (and self-explanatory) configuration file cluster.cnf and adapt it for your needs. To start with, you only have to insert the IP address of this computer.

- Run the Server: `./cluster -ser -new` (the `-new` parameter creates a new empty job queue).

```

MODELS@HOME - The CMBI Distributed Computing Cluster
=====
Written in 2000 by Elmar.Krieger@cmbi.kun.nl
Server IP: 131.174.88.115, Port 1234
Checking for lost jobs, last check at Thu Jan 1 01:00:00 1970
Waiting for clients to connect, Fri May 25 20:42:15 2001

```

- Look at the empty job queue `cluster.job`. This text file should now contain only one header line.

LINUX Supervisor Client

- Supervisor Clients are programs or scripts that have the complete picture of the task "in their mind". Most of the time, these programs are running on the server, otherwise they need NFS access to the cluster directory on the server. How to adapt your script for parallel execution will be explained later. For the moment, run the demo application in subdirectory `muldemo`: This program uses the cluster to calculate the products $1*1$ till $10*10$ like in elementary school ;-). Either run **`muldemo`** (C version) or **`muldemo.py`** (Python).
- Look at the job queue `cluster.job`. This text file should now contain 10 multiplication jobs (explained later). (Note that some "smart" text editors will not show you the changes, because they do not reload the file unless you exit the editor and restart).

LINUX Working Client

- Become root, copy `cluster.zip` to `/usr` and unpack: **`cd /usr; unzip cluster`** . This will create a directory `/usr/cluster` and further subdirectories.
- Change the ownership of `/usr/cluster` to yourself, or whoever will run `Models@Home`
- Edit the central (and self-explanatory) configuration file `/usr/cluster/cluster.cnf` and adapt it for your needs. To start with, you only have to insert the IP address of the Server.

LINUX Working Client with screen saver

- Test that all libraries are in place and installation was successful by running this command:
`/usr/cluster/cluster -tst`

- After 7 p.m. and before 8 a.m., you will see a mainly black screen with one little model (the "night shift" with minimum CPU usage), otherwise you should get the colorful version.
- Edit the file cluster.scr. Here you can choose a different type of screen saver for every client:
 - NORMAL gives you the "true" screen saver.
 - PRUDISH gives you a boring version. Use this one for publically accessible computers, or you will get into trouble sooner or later.
 - BLACK just displays a completely black screen, in case your monitor can detect that and go to power-save mode.
- LOGON gives you a faked Windows NT log on box, if you want to install it on computers where you are not allowed to install it.
- Configure your desktop to run this command as a screensaver, or run it from a terminal for testing first: `/usr/cluster/cluster -cli 1>/usr/cluster/stdout 2>/usr/cluster/stderr &` (In the old days, Models@Home provided its own screensaver to run the command above also when nobody is logged in. The code can still be found in the install_outdated subdirectory (look at 'install.py'), but is unlikely to be functional today, since Linux and Windows have changed a lot since then, and running a custom screensaver at the log in screen is non trivial).
- If you started the Server and Supervisor Client as described above, the 10 multiplication jobs should now be executed.
- Models@Home creates one Working Client for every CPU core in the system. If you want to use fewer cores uses the -cpus command line parameter, e.g. `/usr/cluster/cluster -cli -cpus 5`

LINUX Working Client without screen saver, node in a dedicated cluster

- Run the following command: `/usr/cluster/cluster -cli -con 1>/usr/cluster/stdout 2>/usr/cluster/stderr &`
- If you started the Server and Supervisor Client as described above, the 10 multiplication jobs should now be executed.
- Models@Home creates one Working Client for every CPU core in the system. If you want to use fewer cores uses the -cpus command line parameter, e.g. `/usr/cluster/cluster -cli -con -cpus 5`

WINDOWS Server

- The windows server has not been tested, but you can try to follow the instructions for a Linux server above, with the following differences: The installation directory should be `c:\cluster`, the configuration file is `cluster.cnw`, and the program to run is `cluster.exe`, which may fail to show anyout output due to a mingw32 console issue.

WINDOWS Supervisor Client

- Just follow the instructions for the Linux Supervisor Client. You maybe have to download Python from www.python.org to try `muldemo.py`. Never look at `cluster.job` with Windows Notepad, as this program will not show you any changes unless you exit and restart.)

WINDOWS Working Client

- Log in as administrator, copy `cluster.zip` to `c:\` and unpack with WinZIP or `pkunzip cluster` . This will create a directory `c:\cluster` and further subdirectories. Change the permission of the `c:\cluster` directory so that those users who will run `Models@Home` have write access there.
- Edit the central (and self-explanatory) configuration file `cluster.cnw` and adapt it for your needs. To start with, you only have to insert the IP address of the Server.

WINDOWS Working Client with screen saver

- Follow the instructions for the 'Windows Working Client' above.
- Run `c:\cluster\cluster.exe -tst` to test the installation.
- Check the instructions for the 'Linux Working Client with screen saver' above.
- Configure your desktop to run this command as a screensaver, or run it from a terminal for testing first: `c:\cluster\cluster.exe -cli >c:\cluster\stdout` (In the old days, `Models@Home` provided its own screensaver to run the command above also when nobody is logged in. The code can still be found in the `install_outdated` subdirectory and involved replacing the logon screensaver `c:\winnt\system32\logon.scr` (look at 'install.bat'), but is unlikely to be functional today, since Windows has changed a lot since then, and running a custom screensaver at the log in screen is non trivial, e.g. involving registry keys like `HKEY_USERS\DEFAULT\Control Panel\Desktop\ScreenSave-TimeOut`).

Solving your problems in parallel with Models@Home

As soon as your test-cluster is up and running, and the multiplication job demo has been completed, it time to install your own programs.

Installing your own programs on LINUX Working Clients

- Install the applications you want to run: Create a subdirectory in /usr/cluster for each of them (unless it's a standard application stored elsewhere). Make sure that the program can be run from any other directory. The Exec module will in fact run it from /usr/cluster/jobX, where 'X' is a separate number for each CPU core. The sample application muldemo is already present. When setting up the cluster and creating an initial installation image, it's most convenient to install the programs manually. How to install programs automatically some time later is described further below.
- Edit cluster.cnf: For each of your programs, invent a 6 letter abbreviation (like YASARA, WHATIF, MULDEM etc., called ProgID below) and add a ProgID_RUN entry to cluster.cnf. If the program needs a specific path to be set, you can also add a ProgID_PATH entry (see examples in cluster.cnf).
- If your program contains parts that are expected to change, you should add them to the automatic update list (UPDATES entry in cluster.cnf, individual file names are separated with '|'). If you then place a never version of this file in the updates subdirectory on the Server, it will be automatically transmitted to all Working Clients. The updates subdirectory holds both Linux and Windows related files. If they differ, they must thus also have different names (like cluster.cnf/cluster.cnw. Normally only the executable is different (e.g. "muldemo" for Linux and "muldemo.exe" for Windows)).
- Test your application directly on the Working Client, go to /usr/cluster/job, set your path to exactly the path you specified in cluster.cnf and run it from there: /usr/cluster/ProgSubdir/ProgName -Par1 -Par2 ... The result should have been stored in /usr/cluster/job

Installing your own programs on WINDOWS Working Clients

- Install the applications you want to run: Create a subdirectory in c:\cluster for each of them (unless it's a standard application stored elsewhere). Make sure that the program can be run from any other directory. The Exec module will in fact run it from c:\cluster\jobX, where 'X' is a separate number for each CPU core. The sample application muldemo is already present. At the CMBI we combined Linux compilation and Windows cross-compilation in single scripts, producing executables for both operating systems at the same time (using mingw32, which does not require additional libraries like cygwin). When setting up the cluster and creating an initial installation image, it's most convenient to install the programs manually. How to install programs automatically some time later is described further below.
- Edit cluster.cnw: For each of your programs, invent a 6 letter abbreviation (like YASARA, WHATIF, MULDEM etc., called ProgID below) and add a ProgID_RUN entry to cluster.cnw. If the program needs a specific path to be set, you can also add a ProgID_PATH entry (see examples in cluster.cnw). If your programs opens a window with a specific name, you should add a ProgID_WINDOW entry. This will allow Models@Home to minimize the Window, otherwise it will be visible on top of the screen saver.
- If your program contains parts that are expected to change, you should add them to the automatic update list (UPDATES entry in cluster.cnw, individual file names are separated with '|'). If you then place a newer version of this file in the updates subdirectory on the Server, it will be automatically transmitted to all Working Clients. The updates subdirectory holds both Linux and Windows related files. If they differ, they must thus also have different names (like cluster.cnf/cluster.cnw. Normally only the executable is different (e.g. "muldemo" for Linux and "muldemo.exe" for Windows)).
- Test your application directly on the Working Client, go to c:\cluster\job , set your path to exactly the path you specified in cluster.cnw (set PATH=MyPath) and run it from there: **c:\cluster\ProgSubdir\ProgName -Par1 -Par2 ...** The result should have been stored in c:\cluster\job

Parallelizing your inner loops on the Supervisor Client

- When your applications are ready to be run in parallel, you must create a program that splits the work into individual jobs and sends them to the Models@Home queue. This is usually done by modifying the inner loop of a script/program you already have. Instead of running a program from there, you spawn a Models@Home job to run the program. Sample programs can be found for C (muldemo.c) and Python (muldemo.py) in subdirectory muldemo, they contain the job queue interface, ready to be merged with your own programs.

The individual steps in Python

- Import the cluster interface (cluster.py must be in your working directory):

```
import cluster
```

- Set the path to the Models@Home job queue "cluster.job". In the multiplication example, the Supervisor Client is physically identical with the Server, and the muldemo program is run from subdirectory muldemo. The job queue is thus in the parent directory:

```
# THE JOB QUEUE IS IN THE PARENT DIRECTORY, INSERT ANY OTHER PATH HERE  
joblistname="../cluster.job"
```

- Open the cluster interface (a local job queue called job_list), 'error' is the name of a function handling errors, see muldemo.py:

```
# OPEN CLUSTER INTERACE  
job=cluster.job_list(joblistname,error)
```

- Spawn the jobs. The only interesting point is the job.add method: This adds a job with priority 8 (9 = long jobs, not for screen saver, 8=maximum normal priority, 0=minimum priority) to run MULDEM (that's the ProgID, the client will retrieve the actual command to run from cluster.cn*) with a number of options. These options must be given in job list format and will be described below.

```

# SPAWN 10 MULTIPLICATION JOBS
resultname=[]
for i in range(10):
    resultname.append("result%02d.txt"%(i+1))
    # CLEAR RESULT FILES FROM A PREVIOUS RUN
    if (os.path.exists(resultname[i])): os.remove(resultname[i])
    # ADD THE JOB
    # THE NUMBER TO BE MULTILIED IS GIVEN AS COMMAND LINE PARAMETER TO muldemo.c
    job.add(8, "MULDEM", "%d |"%(i+1)+os.getcwd()+"|||"+resultname[i]+'|')

```

- Append the local job queue to the main queue, jobs now become visible to the Server.

```

# SUBMIT TO MAIN QUEUE AND WAIT FOR RESULTS
job.submit()

```

- Evaluate the results:

```

for i in range(10):
    result=string.atoi(open(resultname[i]).read())
    print "Cluster result for %d*%d: The product is %d.\n" % (i+1,i+1,result)

```

The individual steps in C

are exactly the same as in Python and are described in muldemo.c

The central job queue cluster.job

A job is simply added to the queue by adding a line to the file cluster.job. See the examples in muldemo/cluster.py and muldemo/muldemo.c (search for JOB LIST). In principle this boils down to locking cluster.job by renaming it (so that the server cannot access it while it is being modified), adding the line, and unlocking the file (by renaming it back).

A line in cluster.job is longer than the width of this page, so the following examples first describe the left side, then the right side:

The left side of cluster.job

Here is a typical example:

```
JOB_____S_P_CS_CLIENT IP_____CLIENT NAME_____TIME OF JOB SUBMISSION___CHECK_____  
00000000 W 0  
00000001 A 5 01 123.234.456.789:001 CMBIPC1           Tue May 21 13:46:22 1991 13fed734  
00000002 W 9
```

And here is a description of the fields, from left to right:

- **JOB:** An arbitrary but unique job number with 8 digits, must be assigned when adding a job.
- **S:** Job status (W=Waiting,A=Active), must be 'W' for newly added jobs.
- **P:** Requested priority (0=low, 9=high). The server will submit jobs with higher priority first.
- **CS:** Number of CPU cores used by this job. Normally 01, unless the application benefits from multi-threading.
- **CLIENT IP:** IP address of the Working Client that is currently working on the job, the number of the CPU core on the client is shown after a colon ':'. (Each CPU core can request its own job).
- **CLIENT NAME:** First 16 characters of Working Client name if available
- **TIME OF JOB SUBMISSION:** Time when job was submitted to client. Helpful to check how long a job has been running already.
- **CHECK:** Time of last still alive message received (seconds since 1970, hexadecimal)

The right side of cluster.job

Here is a typical example:

```
PROGRAM__OPTIONS|DIR|COPY|TEXT|RETURN|DONEFLAG  
MULDEM  5 |/elmar/muldem/|||result02.txt||  
WHATIF  |/elmar/whatif|1crn.pdb|STARTUP.FIL="fulchk 1crn.pdb##y#"|pdbout.txt,*.eps|1crn.done  
YASARA  -nss -con -mcr job.mcr |/elmar/yasara|1crn.pdb|job.mcr="LOADPDB 1crn#ADDHYD 1#SAVEPDB  
1,1crnh.pdb#EXITYSR#"|*|
```

And here is a description of the fields, from left to right. Except for 'PROGRAM' and 'OPTIONS', all fields must be separated with a vertical bar '|':

- **PROGRAM:** The six character ID of the program to run, the actual executable path will be looked up by the Working Client in cluster.cnf/cluster.cnw.

- **OPTIONS:** Additional command line options when running the program selected by 'PROGRAM' on the Working Client. (e.g. the data base to blast against, BLAST output format etc.). *IMPORTANT: The last command line parameter must be followed by a <SPACE>.*
- **DIR:** The working directory on the Supervisor Client, where all the data exchange with the cluster takes place, i.e. the Server retrieves data files from there (unless they contain a path) and stores result files there.
- **COPY:** A list of files to copy to subdirectory /usr/cluster/job or c:\cluster\job on the Working Clients, so that they can complete the job. Multiple filenames must be separated with a comma ','. The default location for these files is the working directory specified above with DIR. Filenames can also contain wildcards, e.g. '*' will send the whole content of DIR to the Working Client.
- **TEXT:** A list of text files to copy to subdirectory /usr/cluster/job or c:\cluster\job on the Working Clients, so that they can complete the job. The content of these files is directly inlined as part of the parameters. Look at the WHAT IF example above, where the script STARTUP.FIL is defined: STARTUP.FIL="fulchk 1crn.pdb##y#". This will create a text file called "STARTUP.FIL" in the job directory on the Working Clients:

```
fulchk 1crn.pdb\n
\n
y
\n
```

The crosses '#' thus represent line feeds. If you need double quotes " in your text file, you can choose any other opening and closing character, e.g. STARTUP.FIL=@fulchk 1crn.pdb##y#@. Multiple text files must be separated with a comma ','. The TEXT field is a convenient method to supply job-dependent scripts without having to create a true file for each of them.

- **RETURN:** A list of files to copy from subdirectory /usr/cluster/job or c:\cluster\job on the Working Clients back to the working directory on the Supervisor Client (DIR) as soon as the job is completed. The file return operation is atomic, i.e. if the Supervisor Client detects that a result file is present, this file is also guaranteed to be complete. You can also use wildcards, e.g. '*' will return all the files in the job directory of the Working Client.
- **DONEFLAG:** The name of a file to be created in the working directory of the Supervisor Client as soon as the job is completed. (This file will contain a simple message: "Job completed at

5.00pm,....".) If you only expect one result file, this field can stay empty, as the presence of the result file already indicates that the job has been completed. In some cases, you do however not know how many result files you will get (when using a wildcard in the RETURN field). In these cases wait until the file specified in the DONEFLAG field is present, then you are sure that really all result files have arrived. If you are using the Python interface, you never have to specify this field, the interface will do it for you.

Extending Models@Home to a large network

When your test installation with one server and two clients is working as expected, it is time for the move to the entire LAN. Zip the cluster directories on your two Working Clients and copy them to the other PCs in the LAN, using FTP/Telnet scripts, remote administration or bare feet.

Automatic installation of programs at a later stage

After some weeks of working with the cluster, it will become necessary one day to install new programs on all the clients. There are three ways of achieving that without walking to every Working Client:

- Remote administration tools. For Linux, these can be simple FTP scripts, for Windows other options are available to system administrators.
- "On the fly" installation of small programs: If the program you want to run is small and compact, consists of only a few files and fits into one directory without subdirectories, you can simply transmit it as part of the job data files. So let your job spawning script include all the program files in the COPY field described above, and use "PROGRM" as the program name. If you take a look at cluster.cnf (Linux) and cluster.cnw (Windows), you see that PROGRM simply runs a file called program (Linux) and program.exe (Windows) in the job directory. So with this approach, you can quickly install anything you want, but must transfer the executable with every job.
- Permanent installation of larger programs: If the program is too large or requires files in subdirectories, you must use the update feature to install it. Add all the files including their paths to the UPDATES field in cluster.cnf / cluster.cnw and store them in the update directory on the server (but without creating subdirectories on the server!). In a two pass process, the

Working Clients will first receive the updates for cluster.cnf / cluster.cnw, which tells them which new files to request from the server. These will then be retrieved in a second pass, and the clients will automatically create the required directory tree.

Security issues

In the majority of cases, Models@Home will be used in the intranet only (normally protected by a firewall). Either because this provides already enough computer power, or because the programs run are not freely distributable, or because of the large amount of work required to support individual users in the outside world. If nevertheless the "world wild web" is targeted, the update feature mentioned above should be deactivated, and it is up to the developer to ensure that the executed programs cannot do any damage (e.g. a simple SAVE command in a program script could already overwrite system files in a Windows environment). Also make sure to set the SUBNET_MASK appropriately, otherwise any Models@Home client from outside can "suck" the files in the update directory or job data. Currently, Models@Home does not use encryption.

Useful hints to avoid common pitfalls

- When **testing a new job spawning application**, use a local copy of the job queue "cluster.job" and check that your script really creates valid job descriptions before letting it access the real file. (Garbage in the job queue can give unpredictable results.)
- To **see which clients are working on which jobs**, which jobs are still in the queue etc., just open "cluster.job" with a text editor. (Only under Linux where files are not locked by opening them, and be careful never to save this file back to disc. Also make sure that text wrap around is disabled).
- During program development, it sometimes happens that you spawn jobs, just to find out five seconds later that the job descriptions are bugged, so you want to **remove waiting or running jobs** without stopping the whole server: Rename "cluster.job" to "myname.txt", use a text editor to cut out your jobs, save the file and rename it back to "cluster.job". Never edit cluster.job directly while the Server is running.
- It is also possible to **kill a job that is already running and hanging in an infinite loop** (which will thus never terminate due to a bug in your application): On the server, go to the

current working directory (from where you started 'cluster -ser') and create a file with the name kill_JobNumber (e.g. in Linux, type **touch kill_00001234** to kill job 00001234). The kill will happen when the Working Client sends the next 'still alive' message, which may take up to 30 minutes.

- **If something does not work**, take a look at the file cluster.err. In many cases, you will find a description of the problem there.
- The Windows file systems have a design bug that makes the rename instruction non-atomic. When the Server stores a result file in the working directory of the Supervisor Client, the file is written as cluster.tmp and then renamed to the final name. **There is a short time window when the returned file is present with the final name on disc, but cannot be opened** (i.e. fopen in C returns a NULL pointer, open in Python generates an exception). Take a look at the functions job_loadfilealloc (muldemo.c) or job.loadfile (muldemo.py) to see how this is handled correctly.
- If you add files to the UPDATES entry that are **larger than the memory available on Server** or Working Clients, this will lead to swapping and fail if the file is even larger than the swap space.
- When files are updated via the net, the permission bits are not copied, they all get 0664 by default. Before running an application, the Working Client sets the permission bits of the executable to 0775. However, only the main executable is known to the Working Client (from cluster.cnf/cluster.cnw). So **if your application consists of more than one executable** and is updated via the net, the main executable is responsible for its child programs. (So it must "chmod" them executable before running them).
- If **the Windows screen saver doesn't become active**, make sure that the registry key HKEY_USERS\DEFAULT\ Control Panel\ Desktop\ ScreenSaveActive is set to 1 using regedt32.exe (don't use regedit.exe!!).
- **When running your program on the cluster fails** (you don't get any results back etc.), look in the server output which client last received a job, log on to this client, and go to directory cluster. Check the file cluster.err, then go to the job directory and make sure all the data files are present (have been received from the server). Now set the path you specified in cluster.cnf/cluster.cnw and run the program using the command from cluster.cnf/cluster.cnw with the command line parameters you specified upon job submission. Most of the time, you

can now see where things go wrong.

- **If your client doesn't request jobs from the server**, make sure that there are >200MB free on the harddisk. To increase cluster stability, clients with filled up harddisks don't participate.