

User's Manual

by Elmar.Krieger@cmbi.kun.nl

Date of last revision: 6/05/2003



CMBI, Center for Molecular and Biomolecular Informatics, Toernooiveld 1, NL - 6525 ED Nijmegen, the Netherlands

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.cmbi.nl/models and www.yasara.com/products

Contents

Models@Home License	4
Overview	5
Supervisor Clients	5
Working Clients	5
Server	6
Installation.....	6
The steps to your own cluster.....	7
LINUX Server.....	7
LINUX Supervisor Client	7
LINUX Working Client	8
LINUX Working Client with screen saver.....	8
More details about the screen saver	9
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	11
WINDOWS Working Client without screen saver, node in a dedicated cluster.....	11
Solving your problems in parallel with Models@Home.....	12
Installing your own programs on LINUX Working Clients.....	12
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
Cluster.job example.....	16
Summary of all fields in cluster.job	16
Extending Models@Home to a large network.....	16
Automatic installation of programs at a later stage.....	16
Security issues.....	17
Useful hints to evade common pitfalls.....	17

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.
- 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", submitted (2001) (look at www.cmbi.nl/models for the final reference) and contain the following text: "Models@Home is freely available from www.cmbi.nl/models" with the URL hyperlinked.
- 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.com 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 NT/2000/XP environments and consists of several functional units:

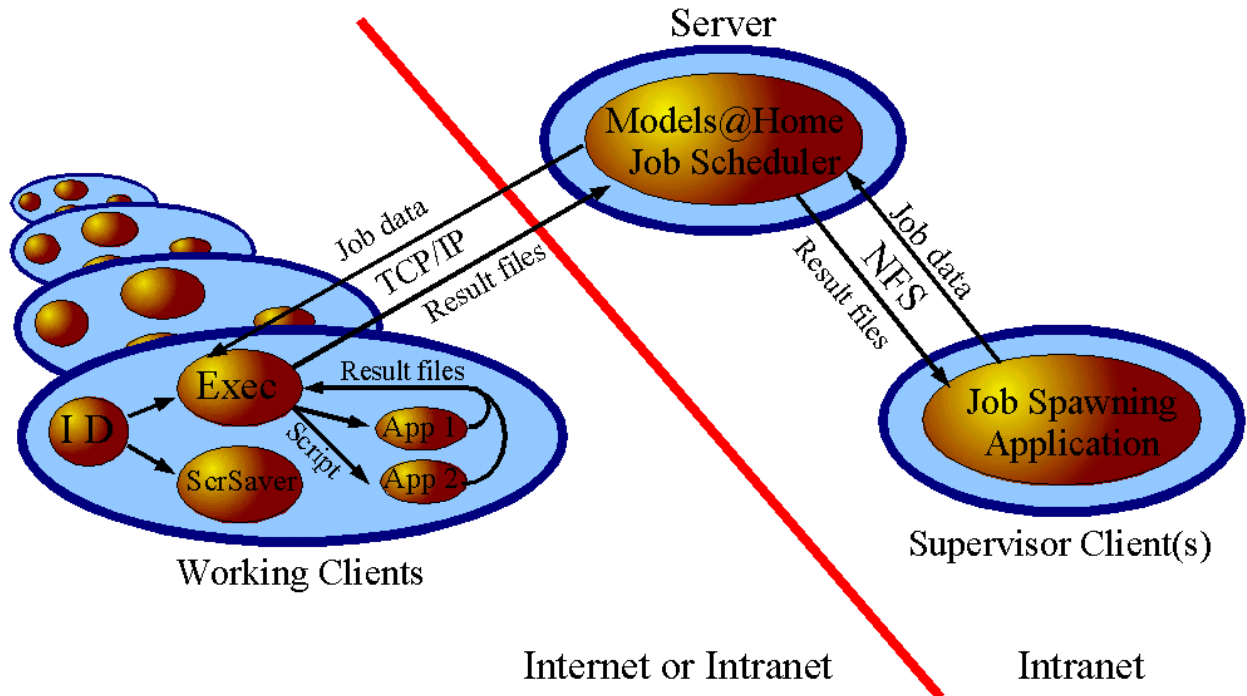


Fig.1: Models@Home data flow. Abbreviations: ID = Idle Detection module, Exec = Program execution module, ScrSaver = graphical screen saver module, App 1+2 = application 1+2, TCP/IP = Transmission Control Protocol / Internet Protocol, NFS = Network File System.

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.com/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").

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 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. If you are using Models@Home in a true cluster, this event can easily be linked to sending an email to the system administrator - Models@Home thus detects when a node fails.

Installation

Models@Home is 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. It has therefore only been tested with these operating systems, any

other platforms are likely to work but will certainly require changes to the source code. Support for Windows 98/ME was added recently by Lionel Caillis. Adapting [Models@Home](#) for MacOS, BeOS or Solaris should also be possible thanks to the platform independent SDL library (www.libsdl.org). Contributions are always appreciated.

The steps to your own cluster

- Choose two to three computers to start with: You need one server (a Celeron 300 works nicely to serve the 30 PCs at the CMBI) and two 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.cmbi.nl/models. You will receive a ~10MB ZIP-archive 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.)
- Go to directory cluster/install and install just the SDL libraries unless you already got them: **cp libSDL* /usr/lib**
- 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.
- Delete to save space: **`rm cluster.zip`**
- 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

- Complete the "Linux Working Client" section and go to installation directory **`cd /usr/cluster/install`**
- See if there is an installation script for your Linux: **`ls install_*`**
- If your Linux is not listed and newer (e.g. RedHat 8.0, Mandrake 9), just run the new all-in-one installation script **`./install.py`** and you are done. (Note that this script has so far been tested with Red-Hat 8.0 only, but should also work with all the others. Suse Linux 8.0 had quite some problems with the login manager `xdm`. If you use `install.py` with Suses >8.0 , you may want to check the "Adding `xdm`" part of `install_suse8`.)
- If your Linux is listed and you use one of the `install_MyLinux` scripts, you also have to: 1) Create a new group: Name **`athome`**, group ID 38475 using for example `Linuxconf` (Red Hat). 2) Create a new user: Name **`models`**, user ID 38475, belongs only to group `athome`, home directory `/usr/cluster`, password: up to you. Make sure that directory `/usr/cluster` now actually belongs to `models.athome`, as some tools forget to adjust owner and group: **`chown -R models.athome /usr/cluster`** Also make sure that directory **`/usr/cluster`** is at least executable for everyone: **`chmod +x /usr/cluster`**. All this is done automatically by `install.py`.
- Test that all the libraries are in place, by going to `/usr/cluster` and running the screen saver: **`./cluster -tst`** . This might fail with an X-error if you are still root (in this case log out and log in as user "models"). 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.
- Restart X (press Ctrl+Alt+Backspace). When the log in box appears, it will take 5 minutes till the screen saver becomes active. The ID module will then copy the file cluster.new to cluster and run **cluster -cli**. This copying is needed to make network update of the executable possible (Linux doesn't allow a program to overwrite its own file with a newer version. Instead cluster.new is updated and cluster exits. The ID module copies cluster.new to cluster and then runs the updated version.)
- If you started the Server and Supervisor Client as described above, the 10 multiplication jobs should now be executed.

More details about the screen saver

In case you are using an exotic Linux distribution and must adapt the installation procedure, here are some hints. During the installation, the following files will be replaced to link the screen saver into the system (backups will be made): /usr/lib/X11/xdm/XSetup_0, /usr/lib/X11/xdm/Xsession, /usr/lib/X11/app-defaults/XScreenSaver, /usr/lib/X11/xdm/Xresources, sometimes /etc/X11/prefdm. In addition, the SDL library will be copied to /usr/lib. With distributions other than Red Hat, SuSe and Mandrake, you will have to look at your versions of the X11 files listed above and probably insert the Models@Home specific changes manually (most of the time this should be done correctly by install.py). By default, the screen saver becomes only active before log in (individual users can then select their own screen savers, as long as they do not need /usr/lib/X11/app-defaults/XScreenSaver). If you want to use Models@Home even when logged in, comment out the **/usr/cluster/scrsavco -exit** instruction at the beginning of /usr/lib/X11/xdm/Xsession.

- **./install_redhat6.1_xdm** installs on Red Hat 6.x. Red Hat 6.x has bugs in the Gnome login-manager (gdm) which crashes X after the screen saver returns, and a bug in the KDE-login-manager that makes it sometimes lose the input focus after the screen saver returns. Using xdm is therefore the only clean solution on Red Hat 6.x.
- **./install_redhat7_kdm** installs on Red Hat 7.x. Red Hat 7.x still has a bug in the Gnome login-manager (gdm) which crashes X after the screen saver returns. The bug in the KDE-login-manager was however solved with KDE 2.0, making kdm the optimum choice (as it's much more comfortable than xdm. You can of course still select Gnome and all other desktops from kdm, only the login-manager is affected).
- **./install_suse7_kdm** installs on Suse 7.x, has however not been extensively tested.
- **./install_mandrake8** installs on Mandrake 8 systems and has been tested only with Mandrake 8.1.

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

- Complete the "Linux Working Client" section and go to directory /usr/cluster/install and type **./install_node.py**

- Create a cron-job that checks every e.g. 5 minutes if **cluster** is running. If not, copy `/usr/cluster/cluster.new` to `/usr/cluster/cluster` and run `/usr/cluster/cluster -cli -con -cd/usr/cluster` . This copying is needed to make network update of the executable possible (Linux doesn't allow a program to overwrite its own file with a newer version. Instead `cluster.new` is updated and `cluster` exits. Your cron-job detects that, copies `cluster.new` to `cluster` and then runs the updated version.)
- If you started the Server and Supervisor Client as described above, the 10 multiplication jobs should now be executed. Working Clients request jobs whenever the previous job is completed or one screen saver cycle is over.

WINDOWS Server

- Open a command prompt and copy `cluster.zip` to any directory and unpack with WinZIP or **pkunzip cluster**. This will create a directory `cluster` and further subdirectories.
- 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 this computer.
- Run the Server: **cluster -ser -new** (the `-new` parameter creates a new empty job queue). Note that we use a Linux Server at the CMBI, so most of the long-time testing was done under Linux only. In addition there is a little "mingw32 feature" that makes the output disappear before it reaches the console, even though `cluster.exe` was compiled as a console application. The status output is therefore currently only available under Linux.
- Look at the empty job queue `cluster.job`. This text file should now contain only one header line.

WINDOWS Supervisor Client

- Supervisor clients are programs or scripts that have the complete picture of the task "in their mind". 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 **python muldemo.py** (Python).
- Look at the job queue `cluster.job`. This text file should now contain 10 multiplication jobs (explained later).
- (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.
- Delete to save space: **del cluster.zip**

- 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

- Complete the "Windows Working Client" section above and go to installation directory: **cd c:\cluster\install**
- Install the package with **install**. During the installation, the default log-on screen saver `c:\winnt\system32\logon.scr` will be replaced (a backup will be stored as `logon.backup`). If you are using Windows 98/ME, the screensavers will be stored in `c:\windows`.
- Test that all the libraries are in place, by going back **cd ..** and running the screen saver: **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.
- Log out again. It will take 15 minutes till the screen saver becomes active. If 15 minutes is too long, you can change the registry key `HKEY_USERS\DEFAULT\Control Panel\Desktop\ScreenSave-TimeOut` using `regedt32.exe` (don't use `regedit.exe`!!). If you have Windows 98/ME or XP Personal, and you are always logged on, then choose the screensaver 'modelsathome' in your display settings. If you have Win98/ME, using the 'logon' screensaver may hang your system due to a Windows bug in thread-management.
- When the screensaver becomes active, the ID module copies the file `c:\cluster\cluster.win` to `c:\cluster\cluster.exe` and runs **c:\cluster\cluster -cli -cdc:\cluster**. This copying is needed to make network update of the executable possible (Windows doesn't allow a program to overwrite its own file with a newer version. Instead `cluster.win` is updated and `cluster.exe` exits. The ID module copies `cluster.win` to `cluster.exe` and then runs the updated version.)
- If you started the Server and Supervisor Client as described above, the 10 multiplication jobs should now be executed. Working Clients request jobs whenever the previous job is completed or one screen saver cycle is over.

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

- As you would need one Windows license per node, I cannot imagine anyone doing that. If this is really necessary, just duplicate the Linux instructions.

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's 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/job. 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 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 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 /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\job. 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 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 pro-

gram 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 ASSUMED TO BE IN THE PARENT DIRECTORY, INSERT ANY OTHER PATH HERE
joblistname="../cluster.job"
```

- Open the cluster interface (a local job queue called job_list):

```
# OPEN CLUSTER INTERACE
job=cluster.job_list(joblistname,error)
(error is the name of a function handling errors, see muldemo.py)
```

- 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 MULTILUPIED 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 returned 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

When adding a job to the queue, beside specifying a priority and the program to run, one must include a number of parameters (the cryptic third function parameter of job.add above). These parameters are contained in a simple text string that is written to cluster.job right after the ProgID. Parameters are separated with '|' and described below:

- **OPTIONS:** Additional command line options when running the program selected by ProgID 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 below, 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 crosses for other purposes, contact us. 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 - resulting in higher performance and less garbage on the disc of your Supervisor Client.

- 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.

Cluster.job example

Left half of the file:

```
JOB NUMB_S_P_CLIENT IP CLIENT NAME TIME OF JOB SUBMISSION CHECK
00000000 W 0
00000001 W 7
00000002 A 8 123.234.456.789 CMBIPC1 Tue May 21 13:46:22 1991 13fed734
```

Right half of the file:

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

Summary of all fields in cluster.job

```
JOB NUMB: An arbitrary job number
S: Job status (W=WAITING,A=ACTIVE)
P: Priority (0=low, 8=high, 9=long job,>12 hours,not for screen saver mode)
CLIENT IP: IP address of client that is currently working on the job
CLIENT NAME: First 16 characters of client name if available
TIME: Time of job-submission to client
CHECK: Time of last still alive message received (seconds since 1970, hexadecimal)
PROGRAM: 6 characters, program to be run on client: WHATIF,YASARA,etc.
OPTIONS: Command line options used when running the specified program
DIR: Working directory on Supervisor Client
COPY: Files to copy to Working Client
TEXT: Text files to copy to Working Client
RETURN: Files to return from Working Client after job is done
DONEFLAG: Filename that will be generated by the Server as soon as job is done.
```

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. It is practically guaranteed that you only have to do that once, because the automatic file update allows changes later, including the Models@Home software itself.

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 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. If the program is large, this may cause a lot of additional network traffic.
- 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 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 evade 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 delete these 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.

- 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 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. Should you need to update such large files, please contact us for the required patch.
- 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.