

# **MicroGrid 2.4.6 User Guide**

**Concurrent Systems Architecture  
Group**

**University of California, San Diego**

**Version 0.46**

Last Revision: Dec 12th, 2004

## Preface

The MicroGrid is a tool which provides the ability to emulate virtual grid infrastructures, enabling scientific study of grid resource management issues. The MicroGrid emulation tool enables repeatable, controllable experimentation with dynamic resource management techniques, a critical activity in the investigation of flexible resource sharing environments posited for future Global Grid scenarios. MicroGrid experimentation complements experimentation with physical resources by providing the capability to emulate both resources not yet deployed (what if scenarios) and rare events such as catastrophic failures. As such, the MicroGrid supports scientific evaluation of adaptive software, resource management algorithms and implementations, and general resource and applications control algorithms over a wide variety of Grid configurations.

For more information, the interested reader is referred to:

[\*Validating and Scaling the MicroGrid: A Scientific Instrument for Grid Dynamics\*](#), Xin Liu, Huaxia Xia, and Andrew Chien, to be appeared in Journal of Grid Computing.

The MicroGrid offers many advantages for grid researchers, such as a tunable process scheduler, pervasive instrumentation for system profiling, and scalability from local to wide area networks.

## Assumptions

Readers should be proficient in the use of a standard UNIX shell, gzip, tar and the build tool GNU make. In addition, general familiarity with the principles of IP networking is helpful. Finally, if you do not understand how to use XML, then developing useful configuration files and network topologies might be difficult, although you should be able to build simple experiments easily.

## Font Conventions Used in This Manual

`Courier` Font is used for:

- UNIX pathnames, filenames, program names, environment variables, user command names and options for user commands
- Anything that would be typed verbatim into code, such as examples of source code and text on the screen.
- MicroGrid tools.
- The contents of XML files, such as document type definitions, elements, symbols (defined constants and bit flags), and macros.

*Italic Courier Font* is used for:

- Arguments to MicroGrid functions, since they could be typed in code as shown but are arbitrary.

*Italic Arial* is used for:

- MicroGrid specific terms.

**Boldface** is used for:

- Chapter and section headings.

## Index

1. Software Requirements
2. Installation
3. Getting Started
4. Experiments without Globus
5. Running an Experiment with Globus
6. Test the MicroGrid Installation
7. Troubleshooting

## 1. Software Requirements

The following tools are required:

- X86 Linux
- GNU make
- GCC 3.x
- Perl 5; with XML DOM module 1.39
- MPICH v 1.2.5.1; with ch\_p4

Optional --

- Globus 2.4.x (If Globus support is required)
- MPICH v 1.2.5.1; with ch\_p4 and Globus device

## 2. Installation

This section provides installation instructions for the MicroGrid emulator with and without support for Globus. Beginning with details about the contents of the package, we later describe how to properly configure the environment, build the application libraries and finally deploy the environment into a user's workspace.

### 2.1 Contents

The MicroGrid package contains:

- massf (network emulator)
  - ssfnet (network simulation protocols)

- dassf (distributed simulation engine)
- wrapper (socket wrapper library)
- sched (processor scheduler)
- MapServer (Virtual IP mapping service)
- control (experiments configuration tool scripts)
- mite2000 (event driven thread pool library)
- globus\_patch (Globus2 patches required by MicroGrid 2)
- build scripts (build and install scripts for MicroGrid 2)
- examples (test programs and application examples)
  - template\_mgrid (template files for non Globus jobs)
  - template\_mgrid\_globus (template files for Globus jobs)
  - topology (Five different virtual Grid configurations)

Download the MicroGrid distribution from the distribution website (<http://www-csag.ucsd.edu/projects/grid/download.html>) and unpack it using 'gunzip' and 'tar'. This will create a directory called 'mgrid2', where you will find example MicroGrid resource files, Globus patches, configuration files, and a number of scripts to build common applications.

## 2.2 Configuration

In order to successfully install and use the MicroGrid, edit the MicroGrid resource file (`mgridrc`) and declare the following variables needed by the build scripts and emulation system.

- `$MGRID_ROOT`: MicroGrid installation path
- `$PERL5LIB`: Path to your installation of Perl5
- `$MPICH4_PATH`: Path to your installed MPICH ch\_p4 device (eg: `/usr/x86-local/mpich/ch_p4`)
- `$MPICHG2_PATH`: Path to your local installation of the MPICH Globus device (eg: `/usr/x86-local/mpich/globus2`)
- `$GLOBUS_BUNDLE_DIR`: Path where the Globus bundles can be found. These source bundles are needed to build MicroGrid enabled Globus services.

Now run `'source mgridrc'` to load the variables into your environment.

If you are a tcsh user, edit and source the <code>mgrid.tcsh</code> file instead of <code>mgridrc</code> .
--

## 2.3 Building the Emulation Libraries

To build the MicroGrid emulation library with Globus support (`mgrid_globus`), enter the `mgrid2` directory and type `'./build_mgrid_globus'`. If your environment settings are correct, `gmake` will begin building the emulator libraries and binaries; be patient the project may take a while to finish.

This script should build and install the mgrid-patched Globus 2 to `$GLOBUS_LOCATION` defined in your `mgridrc` file. Under `$GLOBUS_LOCATION`, you should find the following mgrid-globus specific files:

```
bin/mgridrun
sbin/mgrid-gatekeeper
libexec/mgrid-job-manager
lib/libmgrid.a
```

This MicroGrid enabled Globus installation will not interfere with your original Globus installation (if any), as long as you set the `$GLOBUS_LOCATION` environment variable correctly before using it. See the official Globus website for information regarding Globus installation (<http://www.globus.org>).

If you don't need Globus support, you can build the standalone MicroGrid emulation library (`mgrid`). Enter the `mgrid2` directory and type `./build_mgrid`. Again, be patient since the project may take a while to finish.

## 2.4 User Deployment

Before you may begin using the MicroGrid, it must first be deployed to a shared NFS/SMB filesystem available to all the nodes in your system. To deploy the MicroGrid, first source your `mgridrc` (`mgridrc.tcsh`) file, this updates any variables that may have changed in your `mgridrc` after the initial build. Now run the `mgrid_deploy` script under your `$MGRID_ROOT` directory. This script will deploy your build of the MicroGrid into the directory specified by the `$MG_CONF` environment variable. If this variable has not been declared the `mgrid_deploy` script will create a directory named `./mgrid` in the user's home directory. This directory contains a number of configuration files and other temporary state files the emulator may need at runtime. These files include an updated MicroGrid resource file `mgridrc`, the MicroGrid configuration file `mg_conf`, and a number of internal temporary directories.

After deployment each user must edit the `mg_conf` file (changing each of the variables below) to a unique set of values. If they are not unique bizarre interference may occur.

- `NSE_ADMPORT`: Emulator administration port, through which MicroGrid enabled applications send administrative messages to the emulation.
- `MAP_ADMPORT`: The *MapServer* port, through which the MicroGrid enabled application will add, remove, and query for virtual/real IP address mappings.

- `MAP_ADMADDR`: The IP address of the physical machine, on which the *MapServer* daemon will run.
- `SCH_PORT`: The IP port, through which the command line job dispatch client '*schapi*' will send job activation, termination, and control messages to MicroGrid process Schedulers.

If you plan to use Globus with the MicroGrid, you must find the correct values for the following variables and edit them in `mg_conf` appropriately.

- `LDAP_SERVER`: IP Address or Hostname of the physical machine serving as LDAP server (GIIS).
- `LDAP_PORT`: Port used by the GIIS.
- `LDAP_PASSWORD`: Plaintext of the GIIS password.
- `basedn`: This optional string contains your Globus Domain Name

**See:** `$MASSF_DIR/control/mg_conf.example` for a more verbose example.

## 3. Running the MicroGrid

The MicroGrid allows you to model arbitrary virtual network structures, compute resources, and storage resources. We first explain how a virtual network and resources are described, using examples included with the standard installation. Then, at the end of this section, we explain how the emulator for this configuration is launched.

### 3.1 Building a Virtual Network

First, source the `mgridrc` file in your `$MG_CONF` directory. Now, create a directory where you will store your network configuration files and launch your experiments. For the remainder of this section we refer to this directory as your *Working Project Directory (WPD)*. We recommend that you place your *WPD* in a folder on a NFS/SMB shared filesystem, so that all machines participating in the emulation can get copies of the necessary Virtual Network files. After creating a *WPD*, change to (`cd`) this directory and copy the project template from `$MGRID_ROOT/examples/template_mgrid_globus/*` into your current working directory. If you are targeting a non-Globus application use the files in `$MGRID_ROOT/examples/template_mgrid/*`. You should now have the following files in your *WPD*:

```
MYGRID.dml
machine.dml
MYGRID.phost
MYGRID.vhost
```

## Makefile

Now, begin editing the sample `MYGRID.dml` file and save it to a file named `<GRIDNAME>.dml`. The `<GRIDNAME>` you have chosen will be used to identify your network configuration. Make a note of this `<GRIDNAME>`, as you will use it in the following procedures.

Next, edit the `machine.dml` file in your current *WPD* and list the physical machines available to the network emulator. This file contains a sequence of entries, of the following format:

```
MACHINE [IPADDR hostname ARCHITECTURE os_type PARTITION  
processor_number]
```

The *hostname* field specifies the hostname, or IP address, of a physical host that will run the emulator. The *os\_type*, and *processor\_number* fields respectively, specify the current operating system and the number processors available on the target machine.

→Due to an implementation limitation the *processor\_number* must be set to 1.

→Beware: MicroGrid options used with `mpirun`, do not allow you to include the machine from which you launch the MicroGrid in your `machine.dml` file.

Next, edit the `<GRIDNAME>.phost` file and list the physical machines available to the application. For each physical machine, include one line as below:

```
machine_name CPU_rate CPU_count memory
```

*CPU\_rate* is a relative CPU speed, you can use MHz, FLOPS, or even the ratio to a “standard” CPU.

→Due to an implementation limitation the *CPU\_count* must be set to 1 (only uniprocessor modeling is supported); the *memory* field is not used currently.

Edit the `<GRIDNAME>.vhost` to define the capabilities of each virtual hosts you are going to use. The first line of the file defines the CPU downgrade, the factor by which the emulation rate is slower than wall clock time. For example:

```
cpu_downgrade=5
```

Then, for each virtual host, include one line as below:

```
virtual_name virtual_ID CPU_rate CPU_count memory [gatekeeper]
```

Where *virtual\_name* is the name of the virtual machine used by the user to identify the machine; a "\*" can be used if user doesn't care about the name. The *virtual\_ID* variable is the host ID from topology file. The *CPU\_rate* variable specifies the relative speed of virtual CPU, which should be comparable to the *CPU\_rate* of the physical CPU. The last item, **gatekeeper** indicates the virtual machines where you want to run gatekeeper.

→Due to an implementation limitation the *CPU\_count* must be set to 1; the *memory* field is not used currently.

Now, edit the makefile and set the `$GRIDNAME` variable to the name of your `<GRIDNAME>` file and then execute `make`.

The MicroGrid emulator uses a virtual network topology file that complies with the original SSFNET Domain Modeling Language (DML). You can find more information about the SSFNET DML file format here:

<http://www.ssfnet.org/InternetDocs/ssfnetTutorial-1.html>

Though MicroGrid DML files follow the same format used by SSFNET, they have been modified to include an *emulation\_rate* setting that specifies the CPU scaling factor. This value indicates the rate by which time progresses in the application; Larger numbers result in a slower than real-time rate of progress.

If `make` was able to successfully build your emulated network, you will find the following files in your *WPD*:

- `emulator-LINUX`: The emulator executable.
- `mpirun-emulator`: The script used to launch the emulator.
- `<GRIDNAME>.xml`: The virtual hosts information to be uploaded to the GIIS server.
- `FWTB`: A mapping of the virtual IP addresses in the virtual network to the physical host that provides emulation for each virtual host. At present virtual IP addresses are assigned automatically from the network DML file. You should examine this FWTB file to learn how virtual IPs, and DML addresses, have been assigned.
- `<GRIDNAME>.mapfile`: The physical/virtual mapping information
- `<GRIDNAME>-app.xml`: The configuration used to run application

## 3.2 Launching the Emulator

Congratulations, you are now ready to launch the emulator!

```
% mpirun-emulator
```

This starts the emulator process, and you can now run experiments that communicate through the virtual network. There are a variety of parameters for `mpirun-emulator`; examples of which can be found in `$MGRID_ROOT/template_mgrid/run`.

## 4. Experiments using `mgrid`

This section explains how to instrument simple applications to run under control of the MicroGrid emulator. In this section we assume that you have access to the application's makefile, the compiler object files or equivalent source code. We will begin by explaining how to modify your makefile to install the appropriate hooks in your application. After that, we explain the MicroGrid forwarding table, and how to lookup the identity of a given network node from the contents of the forwarding table. Then we will introduce resource and application configuration procedures, followed by instructions describing how to start the service daemons. This section ends with a description of how to launch your application as a MicroGrid job.

Please skip to section 5 if you plan to use the MicroGrid exclusively with Globus.

### 4.1 MicroGrid Enabling through static linker interception

Now prepare your simple application to run with the `mgrid` by following these three steps.

1.) Modify your application makefile to include `defs.mk` as follows:

```
include ${MASSF_DIR}/wrapper/defs.mk
```

2.) Add `${WRAP_LIST}` to the linker options in your makefile

3.) then delete the preexisting application binary, and rebuild using the makefile.

Your application is now MicroGrid enabled.

You can run original binary codes on the MicroGrid by dynamic binding to the MicroGrid library, please refer to subsection 4.7 for instructions.

### 4.2 Virtual Compute Resources

The forwarding table (*FWTB*) and mapping table (*<GRIDNAME>.mapfile*) provide some useful information about how the virtual resources are mapped on to the physical resources.

The forwarding table contains translations from virtual IP addresses in the emulation to DML node names in the virtual network description (these are not IP addresses). The emulator assigns IP addresses and configures routing tables appropriately, consistent with CIDR hierarchical address management. The *mgrid* system uses the *FWTB* translations to inject/extract application messages to/from the network emulator.

Each line of the *FWTB* file has the following format:

```
IPAddress/Netmask Physical-Node Node-Type DML-Address
```

For example:

```
0.0.1.46/30 machine-23 Router 3:3(1)
```

This entry indicates that the DML node with address *3:3(1)* has been assigned the virtual IP Address *0.0.1.46* with a subnet mask of *30*. This DML node is hosted on the physical node *machine-23*, and is functioning as a router.

There are two Node-Types, router and host. Applications and jobs run only on nodes of type host.

The mapping table contains more detailed information for virtual hosts. Each line of the mapping table has the following format:

```
Virtual-name IPAddress Physical-Node CPU-percentage memory
```

For example:

```
n10.caltech.teragrid.org 0.0.2.40 compute-0-5 32 512
```

This entry indicates that the virtual host *n10.caltech.teragrid.org* is assigned IP address *0.0.2.40* and will run on physical machine *compute-0-5*. The virtual host will use at most *32%* of available CPU cycles and *512MB* memory.

There is another file that is generated: *<GRIDNAME>-app.xml*. This file defines the mapping information in XML format and is used by the scheduler. It includes a sequence of *entry* elements called a *hostmap*.

```
<hostmap>
  <entry>
```

```
<v_host>0.0.1.10</v_host>
<p_host>compute-1-0</p_host>
<speed>90</speed>
<memory>512</memory>
</entry>
...
</hostmap>
```

The *hostmap* example above specifies that jobs for virtual host *0.0.1.10* will be executed on physical host *compute-1-0* and use *90%* of the available CPU cycles and *512MB* memory on that physical host.

### 4.3 Configuring Your Application

Having already configured your virtual resource environment, your next step is to write the job description file that specifies where and how each application process will be executed.

An example file, named *jobs-cs.xml*, can be found in your *WPD* (a copy of this file can be found in *\$MGRID\_ROOT/examples/template\_mgrid*). This file is used to specify a sequence of *job* entities known as a *joblist*.

```
<joblist>
  <stdout>/home/ksmith/tmp/TcpServer.out</stdout>
  <stderr>/home/ksmith/tmp/TcpServer.err</stderr>
  <job>
    <exec>/home/ksmith/tcptest/TcpServer</exec>
    <starttime>0</starttime>
    <param>5234</param>
    <v_host>0.0.1.130</v_host>
  </job>

  <job>
    <exec>/home/ksmith/tcptest/TcpClient</exec>
    <starttime>2</starttime>
    <param>0.0.1.130 5234</param>
    <v_host>0.0.0.34</v_host>
  </job>
</joblist>
```

Each *job* specifies, an application executable, and the number of seconds after the *joblist* is submitted that this job should run. Each virtual host is declared using a *v\_host* tag and command line variables may be specified using a *param* tag.

The example above redirects stdout to */home/ksmith/tmp/TcpServer.out* and stderr to

/home/ksmith/tmp/TcpServer.err. It then starts a *TcpServer* application with a single parameter *5234*. The application is assigned to *v\_host 0.0.1.130*. Finally a *TcpClient* application with parameters *0.0.1.130* and *5234*, on *v\_host 0.0.0.34*, starting 2 seconds after the *joblist* is submitted.

You are almost ready to start your application, but first you have to start the service daemons.

## 4.4 Starting the Service Daemons

To start the MicroGrid service daemons execute the following command.

```
% start_daemon.pl <GRIDNAME>
```

You have now started a *MapServer* on the local machine and a process *Scheduler* on each physical resource defined in the resource XML file (see: Section 4.3.2).

## 4.5 Launching the Application

Congratulations, you can now start your application with the following command.

```
% submit_job.pl <GRIDNAME> MYJOB.xml
```

## 4.6 Launching the Application Manually

For most users, launching the application through the `submit_job.pl` script is the most convenient approach. But if you need more control (for example, you want to debug), you can launch each single process manually by following these steps:

- 1) Login to the machine to execute this process.
- 2) Set environment variable `$MGHOSTNAME` to the IP address of the virtual host for this process. For example, you can

```
% export MGHOSTNAME=0.0.1.20
```

By setting this environment variable, you have turned this shell to a console of the virtual hostname *0.0.1.20*. All `mgrid` jobs executed under this shell will use this virtual IP address as its host IP address.

- 3) Launch the job just as you normally would.

Remember, the MicroGrid Scheduler does not control jobs launched in this manner, as a result such experiments may be somewhat less accurate.

## 4.7 Run application through dynamic binding to the MicroGrid library

You can run original binaries (without relinking to the MicroGrid library) through dynamic binding. Using this method, you can run almost any programs, including Java, Python, etc.

First, define the environment variables “LD\_PRELOAD” and “MGHOSTNAME”:

```
% export
LD_PRELOAD=$MGRID_ROOT/massf/wrapper/libmgrid.so

% export MGHOSTNAME=0.0.1.20
```

Then you can run your binaries as usual. For example:

```
% /bin/hostname
```

will print “0.0.1.20” instead of your physical host name.

← Remember, you must compile the MicroGrid using gcc 3.2 or higher version to use this feature.

<p><b>See:</b> <code>\$MGRID_ROOT/examples/template_mgrid/ld_exec.sh</code> for example.</p>
--

## 5. Experiments using `mgrid_globus`

This section explains how to run a Globus application under the control of the MicroGrid emulator. We begin with a few instructions that describe how to enable Globus for use with the MicroGrid followed by an explanation of the MicroGrid forwarding table, and how to manually lookup a given network node.

### 5.1 MicroGrid Enabling Globus

To prepare your Globus application for use with the MicroGrid, you must perform the following three steps.

1.) Modify your application makefile to include `defs.mk` as follows:

```
include ${MASSF_DIR}/wrapper/defs.mk
```

2.) Add `${WRAP_LIST}` to the linker options in your makefile.

3.) Delete the preexisting application binary, and then rebuild using the makefile.

Your Globus application is now MicroGrid enabled.

## 5.2 Virtual Compute Resources

This section discusses three files that provide useful information about how the virtual resources are mapped to the physical resources: the forwarding table (*FWTB*), mapping table (`<GRIDNAME>.mapfile`), and the application mapping table (`<GRIDNAME>-app.xml`).

If you have already read subsection 4.2 you may skip most of the remainder of this subsection. The only difference is there's a `<gatekeeper>` entry in the application-mapping table.

The forwarding table contains translations from virtual IP addresses in the emulation to DML node names in the virtual network description (these are not IP addresses). The emulator assigns IP addresses and configures routing tables appropriately, consistent with CIDR hierarchical address management. The `mgrid` system uses the *FWTB* translations to inject/extract application messages to/from the network emulator.

Each line of the *FWTB* file has the following format:

```
IPAddress/Netmask Physical-Node Node-Type DML-Address
```

For example:

```
0.0.1.46/30 machine-23 Router 3:3(1)
```

This entry indicates that the DML node with address `3:3(1)` has been assigned the virtual IP Address `0.0.1.46` with a subnet mask of `30`. This DML node is hosted on the physical node `machine-23`, and is functioning as a router.

There are two Node-Types, router and host. Applications and jobs run only on nodes of type host.

The mapping table contains more detailed information for virtual hosts. Each line of the mapping table has the following format:

```
Virtual-name IPAddress Physical-Node CPU-percentage memory
```

For example:

```
n10.caltech.teragrid.org    0.0.2.40    compute-0-5    32    512
```

This entry indicates that the virtual host `n10.caltech.teragrid.org` is assigned IP address `0.0.2.40` and will run on physical machine `compute-0-5`. The virtual host will use at most 32% of total CPU cycles and 512MB memory.

There is another file that was generated: `<GRIDNAME>-app.xml`, which defines the mapping information in XML format and is used by the scheduler. It includes a sequence of *entry* elements called a *hostmap*.

```
<hostmap>
  <entry>
    <v_host>0.0.1.10</v_host>
    <p_host>compute-1-0</p_host>
    <speed>90</speed>
    <memory>512</memory>
    <gatekeeper>6767</gatekeeper>
  </entry>
  ...
</hostmap>
```

This *hostmap* has an entry, which specifies that jobs for virtual host `0.0.1.10` will be executed on physical host `compute-1-0` and will use 90% of the total CPU cycles and 512MB memory on that physical host. If the gatekeeper entry is present, a port number must be specified. The daemon startup script will automatically start a gatekeeper daemon on the indicated virtual machine and port number.

**See:** `$MGRID_ROOT/examples/template_mgrid_globus/camps8-app.xml` for an example.

### 5.3 Application Configuration

To configure your application for `mgrid_globus` you will need to create an XML job description file and a Globus RSL file under your *WPD*.

`Mgrid_globus` uses a new job manager for job submission named `jobmanager-mgrid`. This job manager is comparable to the PBS or Loadleveler job managers that can control multiple virtual hosts. In order to properly use this job manager you must create a separate job description file and

modify the RSL file to dispatch this new `mgrid` job. At present this is the only job manager that we support.

### 5.3.1 `mgrid` Job Description File

Each gatekeeper needs a job description file. You will need to create a job description XML file in your *WPD*. An example file, named `jobs-cs.xml`, can be found in your *WPD*. This file is used to specify a sequence of *job* entities known as a *joblist*.

```
<joblist>
  <stdout>/home/ksmith/tmp/NPB.out</stdout>
  <stderr>/home/ksmith/tmp/NPB.err</stderr>
  <job>
    <exec>/home/ksmith/NPB2.3/bin/mg.A.8</exec>
    <starttime>0</starttime>
    <param>-h</param>
    <v_host>0.0.0.34</v_host>
  </job>
  <!-- ...other job entries... -->
</joblist>
```

Each *job* specifies an application executable; the number of seconds after the *joblist* is submitted when this job should start. Each virtual host is declared using a `v_host` tag and command line variables may be specified using a *param* tag.

The example above redirects stdout to `/home/ksmith/tmp/NPB.out` and stderr to `/home/ksmith/tmp/NPB.err`. It then starts `/home/ksmith/NPB2.3/bin/mg.A.8` with a single parameter `-h`, and the application is assigned to `v_host 0.0.0.34`.

← Remember, if you use multiple gatekeepers, you must have a job description file for each gatekeeper.

**See:** `$MGRID_ROOT/examples/template_mgrid_globus` for examples.

### 5.3.2 The `mgrid` RSL configuration file

The `mgrid` RSL file defines the Globus application to be executed, in a fashion similar to standard Globus RSL files. However, `mgrid` RSL files must use `jobmanager-mgrid` and they include a few special entries.

First, all Globus application must include the service type "jobmanager-mgrid", in the resource string.

If the gatekeeper is running on virtual IP address *0.0.1.10*, listening on port *7549*, and running under user name "*K Smith*", its contact string might look like:

```
resourceManagerContact="0.0.1.10:7549/jobmanager-  
mgrid:/C=US/O=Globus/O=University of California San  
Diego/OU=Computer Science and Engineering/OU=Concurrent  
Systems and Architecture Group/CN=K Smith"
```

You also need to specify the virtual network to be used for this job by using '(project=)' entry. For example, if you use *campus8.dml* as the virtual network configuration file, you need to add:

```
(project=campus8)
```

As usual, use '(count=)' to specify the number of processes launched. And you should also specify the *job\_type* entry. Currently it must be "multiple", even if you only use one running process. For example,

```
(count=4)  
(job_type=multiple)
```

Indicates four instances of the program are going to be launched.

Then it is time to specify your application to be executed. Remember that you have created *mgrid* job description files at section 5.3.2.1, and now they will be used as input files to the MicroGrid jobmanager (*\$MASSF\_DIR/control/submit\_job.pl*). So, in the RSL file, the directory and executable are fixed for your MicroGrid deployment, and the arguments entry is just the job description file. For example, the following entries submit the jobs defined in the *jobs-mg.A.8a.xml* to the gatekeeper.

```
(directory=/home/ksmith/mgrid2/massf/control)  
(executable=submit_job.pl)  
(arguments=/home/ksmith/template2/jobs-mg.A.8a.xml)
```

To sum up, here is an example from which you might want to derive a command line:

```
resourceManagerContact="0.0.1.146:2119/jobmanager-  
mgrid:/C=US/O=Globus/O=University of California San  
Diego/OU=Computer Science and  
Engineering/OU=Concurrent Systems and Architecture  
Group/CN=K Smith")  
(count=4)  
(label="subjob 0")
```

```
(directory=/home/ksmith/mgrid2/massf/control)
(executable=submit_job.pl)
(arguments=/home/ksmith/template2/jobs-mg.A.8a.xml)
(project=campus8)
(job_type=multiple)
(environment = (LD_LIBRARY_PATH
"/home/ksmith/globus2.2/lib") (GLOBUS_DUROC_SUBJOB_INDE
X "0")) (stdout = /home/ksmith/tmp/mgridrun0.out)
(stderr = /home/ksmith/tmp/mgridrun0.err))
```

Remember, `stdout` and `stderr` of `globusrun` are no longer the output of your application (it is the log of the `submit_job.pl`). Instead you should set them in the job description xml file.

See: `$MGRID_ROOT/examples/template_mgrid_globus/campus8.mg.A.8.rsl` for examples.

## 5.4 Daemon Startup

You may now start the service daemons. First launch the virtual network emulation modules that were built in Section 3.3. After they have begun you may start the service daemons.

```
% start_daemon.pl <GRIDNAME>
```

At this point you have started the *MicroGrid MapServer*, *Scheduler*, and the *mgrid-gatekeeper* on the specified virtual hosts.

## 5.5 Submitting Globus Jobs

The `mgridrun` executable is a MicroGrid enabled version of `globusrun`, used to submit Globus jobs, in a fashion similar to `globusrun`. Be sure that you have set `$MGHOSTNAME` appropriately before using this tool. (`$MGHOSTNAME` is used to define the virtual host IP address for any jobs that running under that shell console. See section 4.6 for more details)

### 5.5.1 Submitting Globus Jobs with a RSL File

You may submit jobs using a RSL file, required by Duroc jobs. An example RSL file can be found in `$MGRID_ROOT/examples/template_mgrid_globus/campus8.mg.A.8.rsl`. This RSL file will submit jobs defined in the `jobs-mg.A.8a.xml` and `jobs-mg.A.8b.xml` files.

Example command:

```
% mgridrun -f campus8.mg.A.8.rsl
```

## 5.5.2 Submitting Jobs without a RSL File

To submit a Globus job to the MicroGrid enabled gatekeeper without an RSL file, it is quite similar to that of using a RSL file. You should use `jobmanager-mgrid` and define those entries mentioned in section 5.3.2.1. An example looks like:

```
% mgridrun -o -r "0.0.1.146:2119/jobmanager-
mgrid:/C=US/O=Globus/O=University of California San
Diego/OU=Computer Science and
Engineering/OU=Concurrent Systems and Architecture
Group/CN=K Smith")
(count=4)
(label="subjob 0")
(directory=/home/ksmith/mgrid2/massf/control)
(executable=submit_job.pl)
(arguments=/home/ksmith/template2/jobs-mg.A.8a.xml)
(project=campus8)
(job_type=multiple)
(environment = (LD_LIBRARY_PATH
"/home/ksmith/globus2.2/lib") (GLOBUS_DUROC_SUBJOB_INDE
X "0")) (stdout = /home/ksmith/tmp/mgridrun0.out)
(stderr = /home/ksmith/tmp/mgridrun0.err))
```

## 5.6 Termination

The `kill_job.pl` script provides Job Termination. Example command:

```
% kill_job.pl <GRIDNAME> jobs.xml
```

After your Job has been terminated you may restart it, or shutdown the network emulator using the `kill_emulator` script. Example command:

```
% kill_emulator
```

These scripts must be launched from your current *WPD*.

## 6. Test the MicroGrid Installation

The MicroGrid package comes with 2 example applications that can be used to test the correctness of your installation. One is a simple TCP client/server program to test the `mgrid` installation and the other is a NPB2.3 mg benchmark to test the `mgrid_globus`. You have read through this document before you start testing.

## 6.1 Testing the mgrid installation

To test your installation copy all the files under `$MGRID_ROOT/examples/template_mgrid` to your working directory.

1. Edit the `machine.dml` and `MYGRID-app.xml` to use your local resource
2. Edit the `jobs-cs.xml` to use the correct path name for executable and output files.
3. Build it with `make`
4. Launch the emulator with `mpirun-emulator`
5. Launch daemons with `start_daemon.pl MYGRID`
6. Submit the jobs with `submit_job.pl MYGRID jobs-cs.xml`

This experiment uses the `TcpClient/TcpServer` programs under `$MASSF_DIR/wrapper/tcptest`. The server will service requests from 3 different clients and then quit. If everything goes correctly, you will see the message

```
TcpServer exits successfully!
```

at the end of the server stdout file (`tmp/TcpServer.out.0`)

## 6.2 Testing the mgrid\_globus installation

To test your installation copy all the files under `$MGRID_ROOT/examples/template_mgrid_globus` to your working directory.

1. Edit the `machine.dml` and `campus8-app.xml` to use your local resource
2. Edit the `campus8.mg.A.8.rsl` to use the correct path name for executable and output files.
3. You should also modify the job description files `jobs-mg.A.8a.xml` and `jobs-mg.A.8b.xml` to represent your current path.
4. Build it with `make`
5. Goto the `$MGRID_ROOT/examples/NPB2.3` directory
6. Using `make suite build mg.A.8`
7. Then return to your working directory.
8. Launch the emulator with `mpirun-emulator`
9. Launch daemons with `start_daemon.pl campus8`
10. Submit the job using `mgridrun -f campus.mg.A.8.rsl`

This experiment submits an 8-process mg class-A job to 2 gatekeepers. Each gatekeeper controls 4 virtual machines. If everything goes correctly, you will find the message

Verification = SUCCESSFUL

in the stdout file (tmp/mg.A.8.out.0)

## 7. Trouble Shooting

### 7.1 TODO

In this world, there are too many things that can go wrong. Most of your time is spent diagnosing what's actually not working (Things are going to be added)

1. Memory leak. (nse\_res, nse\_msg, recv buffer msg at Rcv\_Window. Works well with small experimental setups.
2. If `MGHOSTNAME` is not set the application will dump core and the user is not told what is wrong!
2. If the `FWTB` file is missing the application fails with a dump core and the user is not told what is wrong!
3. If the `MG_CONF` file is missing the application fails silently, without telling the user what is wrong!
4. If the user leaves blank lines in the `<GRIDNAME>.phost` file the build will fail without telling the user what is wrong!

### 7.2 FAQ

Q.) Where can I find Perl 5?

A.) See: <http://www.perl.com>

Q.) Where can I find MPICH v1.2.5.1?

A.) <ftp://ftp.mcs.anl.gov/pub/mpi/mpich-1.2.5-1a.tar.gz> ; Archives can be found at <ftp://ftp.mcs.anl.gov/pub/mpi/old/>

Q) What does this error mean:

```
TCPServer::Listen: bind error (107): There is another
application using Port 5494 on this Computer; Please
close the offending program.
p0_11576: p4_error: interrupt SIGSEGV: 11
Killed by signal 2.
```

A) There is a Port conflict. Either you forgot to kill the previous running process or someone else is running an emulator with the same port. To solve this:

- (1) Make sure that you run “kill\_emulator” after you finish the emulation. Please refer to section 4.7 “other tools” for its usage.
- (2) Make sure that different users have different port number assigned

Q.) What does it mean when my virtual network build (Section 3.2) reports:

```
/home/ksmith/MicroGrid/mgrid2/massf/dassf/bin/partition
/home/ksmith/MicroGrid/mgrid2/massf/ssfnet/config/runtime.d
ml LINUX submodel.tmp .machprof mpirun-emulator KNet.dml

make: *** [model.dml] Error 1
```

A.) This error usually means the `$MG_CONF` is not set correctly. Remember, the `$MG_CONF` variable refers to the local user configuration directory, not the `mg_conf` file itself.

Q.) After editing `machine.dml`, and `campus8-app.xml`, I cleaned out the pervious build and I rebuilt and got these errors:

```
/home/x86-local/mgrid2/massf/dassf/lib/libssf-linux-
thread.a(x-ssf-main.kernel.o):
  In function `global_init()':
  /home/x86-local/mgrid2/massf/dassf/src/kernel/ssf-
main.cc:56: undefined
  reference to `__gxx_personality_v0'

/home/x86-local/mgrid2/massf/dassf/lib/libssf-linux-
thread.a(x-cmdline.kernel.o):
  In function `CommandLine::parse(int, char**)':
  /home/x86-
local/mgrid2/massf/dassf/src/kernel/cmdline.cc:540:
  undefined
  reference to `_Unwind_Resume'
  /home/x86-
local/mgrid2/massf/dassf/src/kernel/cmdline.cc:599:
  undefined
```

```
reference to `_Unwind_Resume'  
/home/x86-local/mgrid2/massf/dassf/lib/libssf-linux-  
thread.a(x-cmdline.kernel.o):  
In function `DML_SubModel::operator!()':  
/usr/include/sys/stat.h(.eh_frame+0x12): undefined  
reference to  
`__gxx_personality_v0'
```

A.) This problem indicates that you may have used a version of gcc or mpicc that does not meet our requirements. Make sure that massf is compiled with mpich ch\_p4 and gcc3.0.4.

Q.) When I try to start my application the process returns immediately.

A.) Check your \$mg\_conf directory and make sure it contains a valid MG\_CONF configuration file.

Q.) How do I determine the appropriate emulation\_rate to use in the network DML file?

A.) At present there are no rules limiting your choice. It will depend mostly on your network topology and the expected traffic load. Typical emulation\_rate values range from 5 to 15. When the emulator cannot catch up with the realtime requirement, it will print warning message to the testout-\* log files. You may need to adjust the emulation\_rate setting if you see the following message in the log files:

```
"Warning: Emulator Cannot catch up with the realtime  
network speed. Please increase the emulation_rate int dml  
file."
```

Q.) What should I do if I get the following error when I try to build the emulator in my WPD?

```
~/home/aoo/mgrid2/massf/control/script_conf.pl  
campus8.phost campus8.vhost FWTB campus8.mapfile campus8-  
app.xml
```

Wrong format:

```
/home/aoo/mgrid2/massf/control/script_conf.pl campus8.phost  
campus8.vhost FWTB campus8.mapfile campus8-app.xml
```

Wrong format:"

A.) Check your <GRIDNAME>.phost file to make sure that it does not contain blank lines.

## Appendix A Intercepted function list

The MicroGrid redirects live traffic by intercepting the socket and other I/O related function calls (e.g. system or libc functions) issued by the application. To work correctly, we try to intercept all the network related function calls and predict all possible programming models. In WrapSocket, we have tried our best to capture these function calls, but the list will never be complete. So, whenever MicroGrid is used with a new application, you should first check that the relevant network related function calls are intercepted and test the application. If you find any missing functions please contact the MicroGrid team and we will try to add it to our WrapSocket library.

Accept
Bind
Close
Connect
dup
dup2
execv
execve
execvp
fclose
fcntl
feof
ferror
fgetc
fgets
fork
fprintf
fputc
fputs
fread
freeaddrinfo
fwrite
getaddrinfo
gethostbyaddr
gethostbyaddr_r
gethostbyname

gethostbyname2
gethostbyname2_r
gethostbyname_r
gethostname
getnameinfo
getpeername
getsockname
getsockopt
gettimeofday
IO_getc
IO_putc
listen
main
MPI_Wtime
poll
printf
pthread_create
pthread_exit
read
readv
recv
recvfrom
recvmsg
select
send
sendfile
sendmsg
sendto
setsockopt
socket
uname
usleep
write
writenv