Guide for HHLR Cluster

Stephan Krämer-Eis, Björn Müller

Abstract: This best practice guide gives an overview over the required steps to run BoSSS on the HHLR cluster. It gives some informations about the cluster itself and starts with a step by step tutorial how to install the necessary libraries on the cluster. Furthermore a commented batch script is given to run a BoSSS application on the compute nodes.

1 Notation

$executable The path to the BoSSS binary on your local machine which you want to execute on the cluster

$BoSSSDir Local directory to your BoSSS repository

$TuID Your TU-ID

$home Your home directory on the cluster

$executionDir An arbitrary sub-directory of `$home` to which the executables will be deployed

$databaseDir The location of a BoSSS database on the cluster (usually a sub-directory of `$home`)

$host The host name of one of the login nodes of the cluster. Currently, you can select lcluster2.hrz.tu-darmstadt.de, lcluster3.hrz.tu-darmstadt.de or lcluster4.hrz.tu-darmstadt.de

2 General information about the cluster

The HHLR Lichtenberg cluster consists of 780 compute nodes and 4 login nodes. The cluster is split into 4 sections:

- MPI, for MPI intensive applications
- MEM, for memory intensive applications
- ACC, for applications which are using accelerators, like GPU computations
- SMP, for computation and memory intensive applications (under construction!)

The compute nodes are subdivided into 19 islands: 17 island with each 32 nodes (512 cores), 1 island with 160 nodes (2560 cores) and 1 island with accelerator nodes (CUDA cores and Xeon Phi Coprocessors). For BoSSS applications the MPI section is most suitable, to witch the 17 islands and the island with 160 nodes belong. Each of this nodes consists of 2 processors with 8 cores (in total 16 cores per node) and 32 GB memory.
The file system is divided into 3 sections:

- `/home`: With your application for an account, you get your home directory which you can find under `/home/$TuID`. This directory can be accessed from all nodes and has a daily backup of your data. The quota is limited to 15 GB.

- `/work/local`: these are the local hard drives of the node and can only be accessed from the particular node. Attention: After a job finished, all data will be erased!
- `/work/scratch`: This directory can be accessed from all nodes. With your application, there will be the folder `/work/scratch/$TuID`. You have unlimited quota, but all data are erased after 30 days without any warning!

More informations: `http://www.hhlr.tu-darmstadt.de/hhlr/lichtenberg/index.de.jsp`

3 Installation on the cluster

The HHLR cluster provides the user with some preinstalled libraries, like `openmpi` or `matlab`. The command `module available` lists them all. To run a BoSSS application on the cluster the Linux C# compiler *mono* must be manually installed. Additionally the libraries *ParMETIS* and *HYPRE* are needed. *ParMETIS* is used for partitioning the grid and distributed the cells to each core. *HYPRE* is used to solve large linear systems. This library is needed if you use for example implicit time-stepping schemes.
There are two ways to install the libraries:

- download the latest version from the developers homepage and install them manually
- copy the binaries from the local BoSSS repository onto the cluster

Up to now (December 2013) *mono* and *HYPRE* needs to be installed. Binaries for *ParMETIS* can be found the BoSSS repository.

3.1 Connecting to the cluster

In general, the *ssh* protocol is used to access the cluster. An *ssh* connection be established by simply using the *ssh* command on the console (see the following section), but more sophisticated tools exist that allow for a simpler configuration of advanced connection settings. On Windows machines, the program *PuTTY* is used predominantly, and this tutorial will briefly demonstrate its usage in section 5.4. It should be noted, however, that *PuTTY* is nothing but a graphical wrapper around the plain *ssh* which means that all options can be used on the console, too.

3.2 Installing *mono*

This is a step-by-step guide to install *mono* the cluster. In total it takes about 1 hour to finish the installation!

1. go to the developers homepage `http://download.mono-project.com/sources/mono/` and copy the link of your desired mono version (`$mono-link`).
   As of Oktober 2015 mono-4.2.1 is tested and working.
2. open a git bash and login on one of the four login nodes of the cluster, e.g 1

   ```
   ssh $TuID@$host
   ```

3. create a new directory `mono` in `$home` and open it

   ```
   mkdir mono
   cd mono
   ```

4. download the *mono* archive into the directory

   ```
   wget $mono-link
   ```

5. open *mono* archive

   ```
   tar -xjf mono-x.x.x.tar.bz2
   ```

6. open the created folder

   ```
   cd mono-x.x.x
   ```

7. before compilation, mono has to be configured. Prefix gives the directory for the binaries

   ```
   ./configure --prefix=/home/$TuID/mono/mono-x.x.x_bin
   ```

8. after a successful configuration the compilation can be started

   ```
   make
   ```

   It can happen that the compiler ask for additional prompts. Just type . and return. The whole compilation takes about 50-60 minutes!

9. to finish the installation prompt

   ```
   make install
   ```

10. finally *mono* has to be added to the PATH and LD_LIBRARY_PATH variable. This can be done in the .bashrc file: Go back to your $home directory and open the .bashrc with an editor, e.g. vi

    ```
    cd ~
    vi .bashrc
    ```

    add to following lines for the PATH variable

    ```
    PATH=/home/$TuID/mono/mono-x.x.x_bin/bin:$PATH
    export PATH
    ```

    and for the LD_LIBRARY_PATH variable

    ```
    LD_LIBRARY_PATH=/home/$TuID/mono/mono-x.x.x_bin/lib:$LD_LIBRARY_PATH
    export LD_LIBRARY_PATH
    ```

    Save the changes and reload .bashrc by prompting

    ```
    source .bashrc
    ```

3.3 Installing *HYPRE*

Installing *HYPRE* works similar to the installation routine of *mono* (see section 3.2). Hence only the bash commands are given:

1. go to the developers homepage http://computation.llnl.gov/casc/hypre/software.html and copy the link of your desired hypre version ($hypre-link).

2. login on the cluster

   ```
   ssh $TuID@$host
   ```

3. ```
   mkdir hypre
   cd hypre
   ```

4. ```
   wget $hypre-link
   ```

5. Note the different tar command

   ```
   tar -xzf hypre-x.x.x.tar.gz
   ```

6. `cd hypre-x.x.x/src`

7. Note the additional command –enable-shared

   ```
   ./configure --prefix=/home/$TuID/hypre/hypre-x.x.x_bin --enable-shared
   ```

8. `make`

9. `make install`

10. only the *HYPRE* library is used thus only the `LD_LIBRARY_PATH` variable has to be adjusted in .bashrc

    ```
    cd ~
    vi .bashrc
    ```

    and add

    ```
    LD_LIBRARY_PATH=/home/$TuID/hypre/hypre-x.x.x_bin/lib:$LD_LIBRARY_PATH
    export LD_LIBRARY_PATH
    ```

    save the changes and reload .bashrc by prompting

    ```
    source .bashrc
    ```

3.4 Installing *ParMETIS*

BoSSS does not work with the latest version of *ParMETIS*. Therefore an older version is needed. You can find the binary in the BoSSS repository. Note: You find *ParMETIS* only in the GridOfTomorrow branch!

1. copy the *ParMETIS* binary folder on the cluster

   ```
   scp -r $BoSSSDir/src/ilPSP/layer_0/3rd_party/ParMETIS $TuID@$host:~/ParMETIS/
   ```

2. login on the cluster

   ```
   ssh $TuID@$host
   ```

3. build *ParMETIS*

   ```
   ./ParMETIS/build.sh
   ```

4. add the *ParMETIS* library to the `LD_LIBRARY_PATH` variable in .bashrc

   ```
   vi .bashrc
   ```

   and add

   ```
   LD_LIBRARY_PATH=/home/$TuID/ParMETIS/:$LD_LIBRARY_PATH
   export LD_LIBRARY_PATH
   ```

Note: `export LD_LIBRARY_PATH` needs only to be called once after the last change of the `LD_LIBRARY_PATH`.

Remark: If it does not compile it is easier to use the compiled binaries instead. You can find them in `\\fdyprime\misc\HHLR_binaries\ParMETIS\bin`

3.5 Installing *PARDISO V5*

1. download the newest *PARDISO* Version from the Website *http://www.pardiso-project.org* or from `\\scratch\kummer\pardiso.zip` and copy the `.so`-file to `\home\$TUID/PARDISO`
2. add the *PARDISO* library to the `LD_LIBRARY_PATH` variable in .bashrc

   ```
   vi .bashrc
   ```

   and add

   ```
   LD_LIBRARY_PATH=/home/$TuID/PARDISO:$LD_LIBRARY_PATH
   export LD_LIBRARY_PATH
   ```

3. get a licence from the pardiso website
4. add the following to your xml-file

   ```
   <sparsesolver name="PARDISO-V5">
     <type>direct</type>
     <library>pardiso</library>
     <specific>
       <Version>v5</Version>
       <!-- username '$TUID', (Lichtenberg) no NODE-lock : -->
       <LicenseCode>PUT LICENCE CODE HERE</LicenseCode>
     </specific>
   </sparsesolver>
   ```

5. enable *OpenMP* parallelization in the job file (see examples)

3.6 Testing the libraries

All necessary libraries are installed on the cluster. Now we can test them by running a small BoSSS application. As test problem, we choose the ipPoisson problem. In this particular case we run the problem directly on the login node, because it takes only a few seconds to compute. In general: Never run a computation on a login node! How to run a computational heavy application is explained later in section 5.

1. compile the ipPoisson project (in src/public/L4-applications/ipPoisson) on your local machine, e.g in Visual Studio
2. transfer the application and its dependencies to the cluster by using the `bcl deploay-at` command (`$executable` is for example `/c/BoSSS/src/public/L4-applications/ipPoisson/bin/Release/ipPoisson.exe`)

   ```
   bcl deploy-at $executable sftp://$TuID@$host:~/ipPoisson
   ```

3. `bcl deploy-at` only copies the application but not a control file. Copy a suitable control file by using `scp`

   ```
   scp control-example.xml $TuID@$host:~/ipPoisson
   ```

4. login on the cluster

   ```
   ssh $TuID@$host
   ```

5. before running the ipPoisson problem three additional libraries needs to be loaded: `acml`, which includes the `BLAS` library and `openmpi`. For running `acml`, the module `gcc` is additionally needed.

   ```
   module load acml
   module load gcc
   module load openmpi/gcc/1.6.5
   ```

6. now the program can be started

   ```
   cd ipPoisson
   mono ipPoisson.exe --control control-example.xml
   ```

7. if everything is working, the program should be end with `converged? true`. Then a parallel run on two cores can be tested:

   ```
   mpiexec -n 2 mono ipPoisson.exe --control control-example.xml
   ```

If both tests run, the installation of BoSSS on the HHLR cluster was successful.

4 Setup and synchronization of the BoSSS database

For most of the BoSSS applications you need a database (db). Since you cannot use the graphical Database Explorer on the cluster, you need to synchronize your db on your local machine in order to evaluate your results.

4.1 Setting up a database

The easiest way to setup a database on the cluster is to generate it locally and then just copy it onto the cluster.

1. create locally a new folder for your db, e.g. `c:\BoSSS_db`
2. open git bash and go to the folder, then prompt

   ```
   bcl init-db
   ```

3. now you have created a new db locally. By using `scp` you can copy it to the cluster:

   ```
   scp -r /c/BoSSS_db/ $TuID@$host:~/
   ```

4.2 Synchronizing databases

There are many ways to synchronize the db. An easy way is to use the tool *WinSCP*. First you have to enter your login data, shown in figure 1. After logging in, you see on the left side your local file system and on the right side is your `$home` directory on the cluster. Browse in both windows to the generated db (according to section 4.1 this would be `c:\BoSSS_db` on your local machine and `/home/$TuID/BoSSS_db` on the cluster). *WinSCP* has a build in synchronization tool, which is indicated with a red circle in figure 2. For the synchronization you have different options, e.g. from local to remote, from remote to local or in both directions. You can also copy single files per drag and drop between both windows.

4.3 Tips and Tricks

- As mentioned in section 2 you have a quota of 15 GB in your home directory. If your planning big and/or many simulations, you might exceed this limit. Instead of using

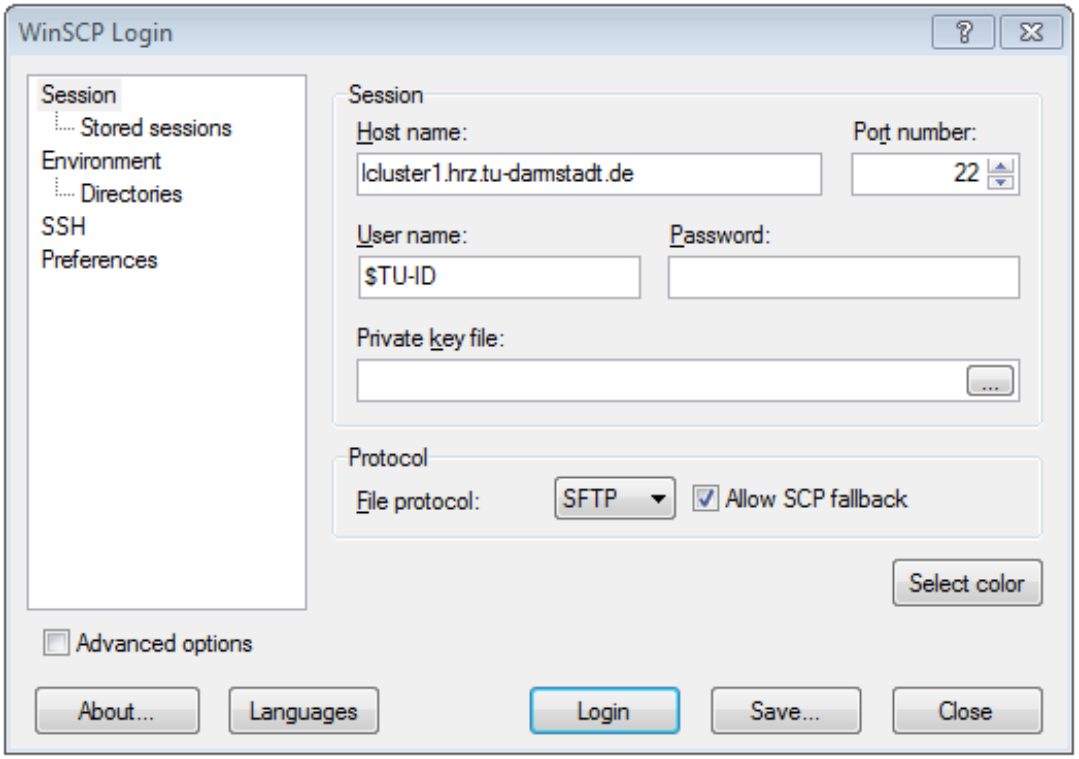


Figure 1: WinSCP - Login data

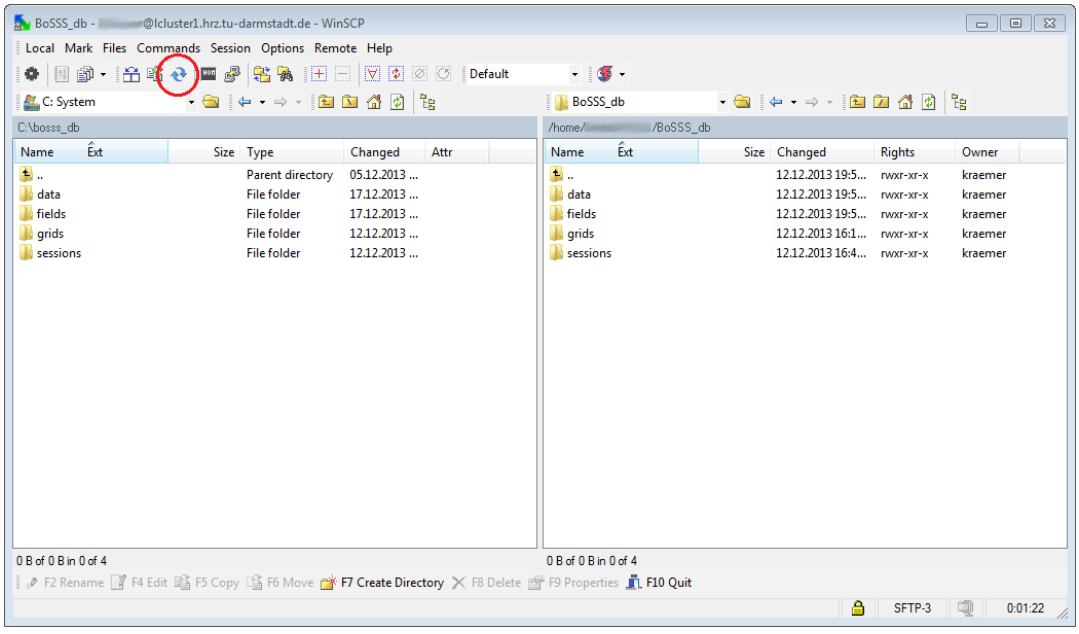


Figure 2: WinSCP - Synchronization

your `$home` directory, you can use `/work/scratch/$TuID` for your db. It has the advantage that you have nearly unlimited quota and `/work/scratch` is a faster file system, i.e. less time is needed to save each timestep. But be careful: You have to take care of the data. On `/work/scratch` data is deleted after 30 days without any warning!

- If you have big databases and want them to synchronize only in one direction, e.g from the cluster on your local machine, it can take some time. Reason is that BoSSS saves every field in each timestep in a single file, hence you have lots of small files to copy. It can be faster to put the whole db in one archive and then just copy the archive to your local machine.

5 Working on the HHLR Cluster

As mentioned in Section 2 the cluster has 4 login nodes and 780 working nodes. The login nodes are only intended to access the cluster, to copy data and to prepare/submit jobs. Do not start any computation on the login nodes!

5.1 Prepare a job

The preparation for a job is similar like the first steps of testing the libraries in section 3.6:

1. compile your application on your local machine in Visual Studio
2. transfer the application and its dependencies to the cluster

   ```
   bcl deploy-at $executable sftp://$TuID@$host:~/$executionDir
   ```

3. optional: Use mono's Ahead of Time (AOT) compilation feature[1] to gain some performance. To that end, navigate to `~/$executionDir` and execute the following command:

   ```
   mono --aot=full -O=all *.exe *.dll
   ```

   This will precompile your code with additional optimizations that are not (fully) available during runtime. The actual speed-up strongly depends on the application, but is usually in order of 4% to 7%.

4. again the control file needs to be copied by using `scp` or *WinSCP*

   ```
   scp control-file.xml $TuID@$host:~/$executionDir
   ```

5. do not forget to adjust the path of your database in your control file. Attention: Use always the full path, e.g.

   ```
   /home/$TuID/$databaseDir/
   ```

   or

   ```
   /work/scratch/$TuID/$databaseDir/
   ```

5.2 Create and submit batch-script

Now all necessary files are on the cluster. The HHLR cluster uses the LSF batch system, which calculates when and on which node the job can be started. First you have to create a batch-script, which then will be submitted. The nodes executes the batch-script line by line. The

[1] `http://www.mono-project.com/AOT`

script consists of two parts: a “Head” part where all informations for the LSF system are specified and a “body” where the commands are listed.

Listing 1: Batch-script: head

```
#!/bin/sh
# Job name
#BSUB -J $job-name
#
# File / path where STDOUT will be written, the %J is the job id
#BSUB -o /home/$TuID/$executionDir/$job-name.out%J
#
# File / path where STDERR will be written, the %J is the job id
#BSUB -e /home/$TuID/$executionDir/$job-name.err%J
#
# Request the time you need for execution in minutes
# The format for the parameter is: [hour:]minute,
# that means for 80 minutes you could also use this: 1:20
#BSUB -W 00:00
#
# Request virtual memory you need for your job in MB
#BSUB -M 1000
#
# Request the number of compute slots you want to use
#BSUB -n 2
#
# Specify the MPI support
#BSUB -a openmpi
#
# Specify OPENMP support (for Pardiso only)
# Specification of OMP_NUM_THREADS is done automatically
# by LSF/Operating System
#BSUB -a openmp
#
# Specify your mail address - for activation replace <your-name>
# and remove prepending "# "
# #BSUB -u <your-name>@fdy.tu-darmstadt.de
#
# Send a mail when job is done - for activation remove prepending "# "
# #BSUB -N
```

In listing 1 the head of an example script is shown. With `BSUB` specifications for the LSF queue are made. Important are `BSUB -W` and `BSUB -M`: If the requested time or memory consumption is reached the jobs ends, weather your BoSSS application finished or not. In listing only the important `BSUB` commands are used. More informations are given in the manual page `man bsub`.

Listing 2: Batch-script: body

```
# Loading the required module
module load gcc
module load openmpi/gcc/1.6.5
module load acml
# Give an overview about all load modules
module list

mpiexec -n 2 mono /home/$TuID/$executionDir/$executable ...
        ...--control /home/$TuID/$executionDir/control-file.xml
```

Listing 2 are the commands which are executed. First the needed libraries are loaded. `mpiexec` starts the BoSSS application. `-n 2` gives the number of requested cores. It must be the same number like `BSUB -n 2` in the head of the batch script. To submit the batch-script to the queue, prompt

```
bsub < batch-script.sh
```

The batch-script is attached to this document.
Note: If you are using the control-file V2 you have to use it like this

```
... --control 'cs:$ExampleControlfile'
```

5.3 Useful LSF commands

- **bjobs** gives an overview of all your waiting and running jobs and their `$job-ID`
- **bpeek [-f] job-ID** shows the standard output of the running job with `$job-ID`
- **bkill job-ID** kills the job
- **bhosts** gives an overviews of utilized capacity of the different compute island

5.4 Using the database explorer on the cluster

Most functions of the databases explorer v2 (*DBEv2*) should work on the cluster just like the do on your local machine. The most important exception is the export of timesteps in the *TecPlot* and *CGNS* formats. In order to install the *DBEv2* on the cluster, you may simply execute the command

```
bcl deploy-at $BoSSSDir/bin/Release/DBEv2 sftp://$TuID@$host:~/DBEv2/
```

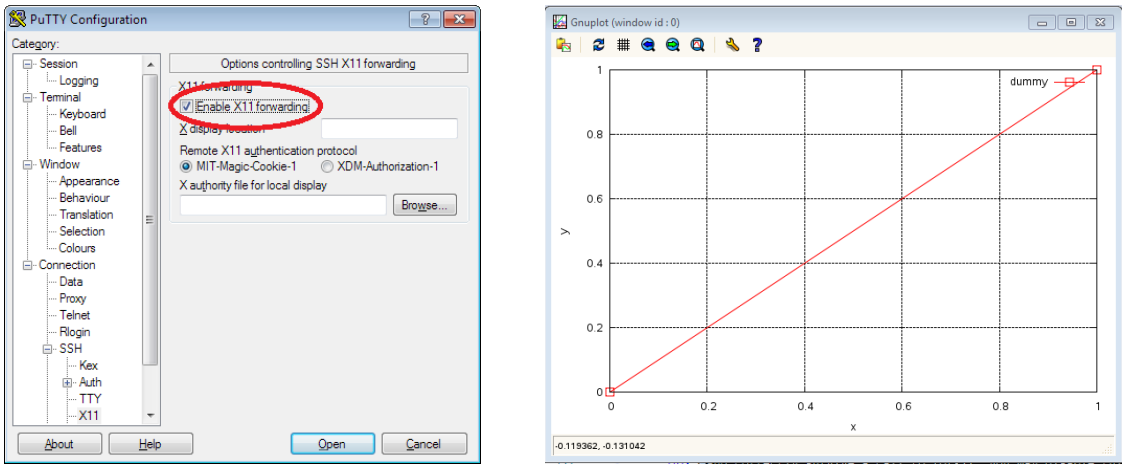
on your local machine. After establishing a connection to the cluster, you can start the *DBEv2* via

```
mono DBEv2/DBEv2.exe
```

Obviously, the list of databases will be empty on the first start, since the location of the databases on the cluster has not been configured yet. In order to do so, you have to create a database configuration file at `~/.BoSSS/etc/DBE.xml` by hand. The content of the file should be given by

```
<?xml version="1.0" encoding="utf-8"?>
<DBEControl>
  <Databases>
    <Database>
      <path value="/home/$TuID/bosss_db" />
    </Database>
  </Databases>
</DBEControl>
```

if your database is located at `~/bosss_db`. When you start the *DBEv2* the next time, this database should be loaded automatically.



(a) Checkbox (b) Exemplary output

Figure 3: PuTTY - Forwarding of graphical output

Some parts of the *DBEv2* create graphical output (e.g., when plotting the residuals). In order to be able to use this feature on the cluster, you to forward graphical output to your local machine. This feature is called *X11* forwarding and is activated by establishing the connection via

```
ssh -X $TuID@$host
```

or by checking the corresponding in *PuTTY* (see Figure 3a). Moreover, you need a software that is able to render the graphical output from the cluster on your local machine. To that end, you have to make sure that the tool *Xming* is running when connecting to the cluster. The tool should already be installed on all machines at the fdy. After starting, it automatically minimizes to the system tray and waits for input to render.

You can test your setup by starting the *DBEv2* on the cluster and running the command

```
new DataSet(new double[] { 0, 1 }, new double[] { 0, 1 }, "dummy").Plot()
```

which should generate the graph displayed in Figure 3b on your local machine.

5.4.1 Troubleshooting

- **MPI-Error**: The *DBEv2* uses MPI communication, hence the MPI module needs to be loaded before starting the *DBEv2* via

```
module load openmpi/gcc/1.6.5
```

- **Library missing**: After deploying the *DBEv2*, the DBEv2 folder contains a file *Mono.CSharp.dll*. It can be that the *DBEv2* doesn't find this library, because it searches for *Mono.Csharp.dll*. Just copy the file:

  ```
  cp Mono.CSharp.dll Mono.Csharp.dll
  ```

- The **git-bash** on windows machines causes some problems after starting the *DBEv2*, i.e. only a black display appears and the program crashes. So far only the *DBEv2* only runs with *PuTTY*.