

Tutorial:
**Desktop Grid installation &
application development using
DC-API**

Gábor Gombás
MTA SZTAKI

Outline of the Process

0. **Installing the base operating system: Debian Sarge**
1. Basic configuration of the operating system
2. Installing the SZTAKI LDG packages
3. Creating a sample BOINC project
4. Writing a sample application using the DC-API
 - Client
 - Master
5. Deploying the sample application on the LDG
6. Trying it out

What Will Be Needed

- **Basic knowledge about the Linux command line (shell)**
- **Basic C programming knowledge (nothing serious)**
- **What we will **NOT** do:**
 - **System management**
 - **BOINC administration**

Step 0: The Basic Operating System

- **The tutorial uses virtual machines where the installation of the operating system differs significantly from a normal install so this step is not part of the tutorial**
- **The SZTAKI LDG builds on the Debian/GNU Linux distribution**
- **The currently supported version is 3.1 codename Sarge**
- **Upgrade to 4.1 (Lenny) is planned but the installation/configuration steps will be very similar**

Step 1: Configuring the Basic Operating System

- Log in to the pre-installed virtual machines. Open a terminal window and type:

```
ssh root@ldgNN.terem
```

– Password: *ldg*

- Edit **/etc/apt/sources.list** and add the following (in one single line):

```
deb http://www.desktopgrid.hu/debian/ sarge lpds
```

- Editors available on the **ldgXX** machines: **vim**, **jed**, **joe**, **mcedit**

Step 1: Configuring the Basic Operating System

- Refresh the package cache:

```
aptitude update
```

- Add a new user who will be the project administrator

```
adduser pradmin
```

Select a password that you can remember

Step 2: Installing the LDG Packages

SZTAKI LDG Packages:

BOINC installation made easy

Step 2: Installing the LDG Packages

- Install the dependencies:

```
aptitude install apache2-mpm-prefork
```

```
aptitude install libapache2-mod-auth-plain
```

```
aptitude install libapache2-mod-php4 php4-cli
```

```
aptitude install mysql-server-4.1 php4-mysql
```

- On a production machine you **SHOULD** set the MySQL root password to something non-empty
- And now the “real” stuff:

```
aptitude install szdg-local-server boinc-dev
```

```
aptitude install libdcapi-boinc-dev
```

Step 3: Creating a BOINC Project

- Creating a new BOINC project takes just a single command:

```
boinc_create_project --name=test --long-name=Test
```

- This will create a UNIX user named **boinc-test** and a MySQL database/user named **boinc_test**
- All files belonging to the project are under the directory **/var/lib/boinc/test**
- Make the **pradmin** user a project administrator:

```
boinc_admin --name=test --add pradmin
```

- The password asked by **boinc_admin** is for the web interface

Step 3: Creating a BOINC Project

- The project is now ready to use
- Root privileges are not needed anymore, so log out as root and log in as **pradmin**
- You can become the project administrator by running

```
sudo su - boinc-test
```

- After the above command the environment is set up so that you can issue BOINC administrative commands directly, such as

```
start
```

Step 3: Creating a BOINC Project

- Now the project is up and running. It is available at

<http://ldgNN.terem/test>

The full BOINC documentation is available at

<http://boinc.berkeley.edu>

Step 3: Creating a BOINC Project

- A small detail before we continue: edit **~/project/config.xml** and remove the following line:

```
<one_result_per_user_per_wu/>
```

- This setting is good for a real project, but can make testing harder

Step 4: Writing an Application

- Writing an application from scratch would be too time consuming, so we have provided the starting ground
- You can find a prepared skeleton in **[primesearch-demo-5.01.tar.gz](#)**
 - Example code: counting primes
 - Build framework for DC-API
- The example code runs on a single machine, your task will be to adapt it to DC-API
 - The code is already organized in a way to make that easier :-)
 - In real life you usually have to refactor the existing application code somewhat

Try the Application First

- Copy the archive to the DG machine

```
scp primesearch-demo-5.01.tar.gz  
pradmin@ldgNN.terem:
```
- Log in and extract the archive

```
tar xzf primesearch-demo-5.01.tar.gz  
cd primesearch-demo-5.01
```
- Configure and build the application

```
./configure --host i686-pc-linux-gnu --build i686-  
pc-linux-gnu  
make
```
- Run it

```
cd src  
echo 100 > master.input  
./primesearch-master
```

The Build Environment

- The example package uses **autoconf** & **automake**
- The DC-API libraries use the **pkg-config** tool from **freedesktop.org** for providing the settings needed for compilation & linking
- There are some macros in **cf/dcapi.m4** that allow selecting the DC-API backend (flavor) and disabling/enabling the building of the master and client side
- The sources we are going to work with are in the **src** subdirectory
- You don't have to touch the build environment if you don't want to :-)

Application Structure: Client

- Let's start with the client application: **src/client.c**. There are 3 functions:
 - **is_prime()** does the prime-testing
 - **process_input()** reads the input file, calls **is_prime()**, and creates the output file
 - **main()** is the main program, it just calls **process_input()**
- What needs to be done:
 - Include the DC-API client header at the beginning
 - Add DC-API calls on application start & exit
 - Call DC-API functions for translating file names

DC-API Documentation

- The DC-API documentation is available at <http://www.desktopgrid.hu>
 - Select “Document Folder”/”Documentation” from the side menu
- What we need is the “Reference” section at the bottom of the page

Preparing the Client Application

- Include the DC-API client header just after the other headers

```
#include <dc_client.h>
```

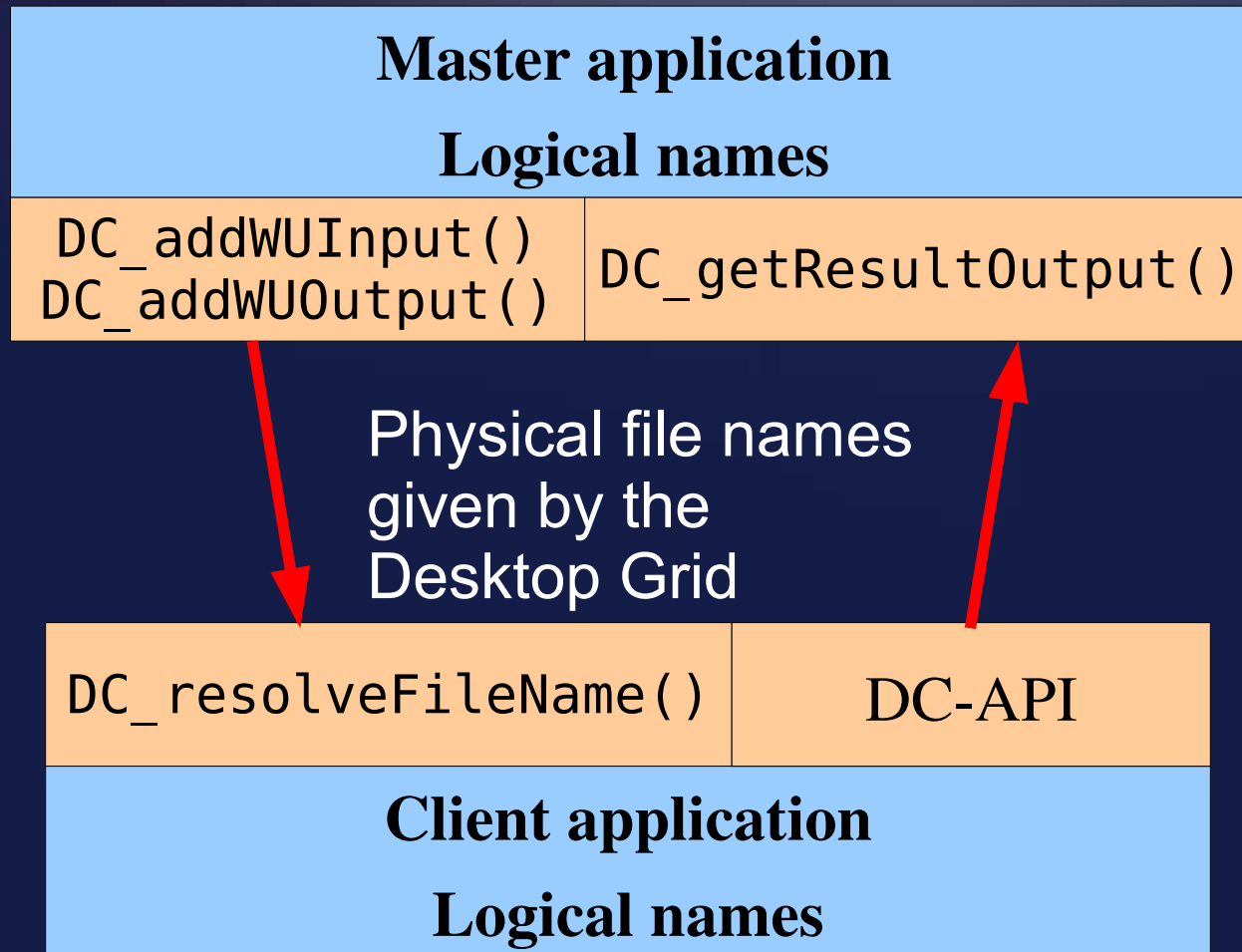
- Initialize the DC-API at the beginning of `main()`

```
ret = DC_initClient();
```

- Instead of `exit()`, we must call `DC_finishClient()`

```
DC_finishClient(ret ? 1 : 0);
```

File Name Translation



Preparing the Client Application

- The **DC_resolveFileName()** function performs the file name translation from logical names used by the client to physical names used by the middleware
- It takes two arguments:
 - The file mode – input, output or temporary
 - The logical file name
 - The special value **DC_CHECKPOINT_FILE** is used for reading/writing the checkpoint image

Preparing the Client Application

- We must do the logical → physical file name conversion in `process_input()`

```
static int process_input(const char *input_name,  
                        *output_name)  
...  
    char *input_file, *output_file;  
  
    input_file = DC_resolveFileName(DC_FILE_IN,  
                                   input_name);  
    output_file = DC_resolveFileName(DC_FILE_OUT,  
                                    output_name);
```

Preparing the Client Application

- Build the client application using **make**
- Try running the master again, it should still work
 - Now the client is linked with BOINC so it will create some extra files

Application Structure: Master

- We have the following functions in the example:
 - `read_global_input()` reads the search limit
 - `process_result()` parses a result file generated by the client
 - `run_prime_test()` tests if a given number is a prime
 - `main()` connects it all

Global variables:

- `limit` is the search limit
- `found` contains the number of primes found so far
- `errors` contains the number of failed prime tests – if it is non-0, the end result may not be correct

Preparing the Master Application

- **“Gridification” of the master is relatively easy:**
 - **The prime tests are independent so they can be sent to independent clients simultaneously**
 - **The order the tests complete does not influence the result**
 - **Instead of local execution, we have to send the client application to the Desktop Grid**

Preparing the Master Application

- Include the DC-API client header just after the other headers

```
#include <dc.h>
```

- Initialize the DC-API at the beginning of `main()`

```
char *config;  
if (argc > 2) config = argv[1] else config = NULL;  
ret = DC_initMaster(config);
```

- Set the callback function for processing finished results

```
DC_setMasterCb(process_result, NULL, NULL);
```

Preparing the Master Application

- **process_result()** needs to be modified to be used as a callback:

```
static void process_result(DC_Workunit *wu,  
    DC_Result *result)  
{  
    ...  
    char *filename;  
    if (!result) { errors++; return; }  
    filename = DC_getResultOutput(result, "out.txt");  
    ...  
    free(filename);  
}
```

Preparing the Master Application

- Now, we have to modify `run_prime_test()` to submit the job to the DG. Instead of calling `system()` and `process_result()` directly, we now do:

```
DC_Workunit *wu;
```

```
wu = DC_createWU("primesearch", NULL, 0, NULL);
```

```
DC_addWUInput(wu, "in.txt", "in.txt",  
              DC_FILE_VOLATILE);
```

```
DC_addWUOutput(wu, "out.txt");
```

```
DC_submitWU(wu);
```

```
submitted++;
```

Preparing the Master Application

- We need a new global variable to count the running WUs

```
static int submitted;
```

- At the beginning of `process_result()` we have to also update `submitted`:

```
--submitted;
```

- `main()` must poll for the results, so after the loop calling `run_prime_test(i)` add

```
while (submitted) DC_processMasterEvents(30);
```

Preparing the Application

- If all done, do

```
make package
```

```
cp primesearch.tar.gz /tmp/
```

- This will create a file named **primesearch.tar.gz** that contains all the executables and XML descriptors for the deployment script
- You can see how the descriptors look like, they can be found as **primesearch-client.xml** and **primesearch-master.xml** in the **src** directory

Step 5: Deploying the Application

- Log in (if not already logged in) and become the project administrator:

```
ssh pradmin@ldgNN.terem  
sudo su - boinc-test
```

- In standard BOINC, application deployment is a many-step process.
- With the SZTAKI LDG packages, it is just one command:

```
boinc_appmgr --add /tmp/primesearch.tar.gz
```

Step 5: Deploying the Application

- What **boinc_appmgr** does are:
 - Places the client executables where BOINC expects them and optionally signs the executables
 - Registers the client executables in the **project.xml** file and in the BOINC database
 - Installs the master application in a new directory
 - Modifies the DC-API configuration of the master to match the project's settings
 - Adds the master application to **config.xml** so it is automatically started/stopped by BOINC when the project is started/stopped
 - Also configures a validator, in our case it will be the **sample_trivial_validator** that just accepts everything

Step 5: Deploying the Application

- We need an input file for the master:

```
echo 20 > ~/master/primesearch/master.input
```

- The project must be restarted after installing an application:

```
stop
```

```
start
```

Now the application (and the project) is up and running.

Step 6: Try It

- Create a new user account by visiting the homepage of the project at: <http://ldgNN.terem>
- ...and selecting 'Create account' and following the on screen instructions
- As a project admin extract the database password from the `~/project/config.xml` file and log in to the mysql database, by executing the 'mysql' command
`mysql boinc_test`
- In the mysql console execute the following command to retrieve your account key, which you will need to attach to the server with a client computer (or enter a valid e-mail address and it will get mailed to you):
`select authenticator from user;`

Step 6: Try It

- Start the BOINC client on your desktop with the following command:

```
boinc_client --attach-project  
http://ldgNN.terem/test <auth>
```

Enhancement of the Client I.

- If the application will run on non-dedicated machines, the client should report the completion ratio, so the user sitting in front of the machine knows the state of the workunit
- `DC_fractionDone()` can be used to report the ratio of the work completed. So inside the loop in `is_prime()`:

```
DC_fractionDone(sqrt(number) / (double)i);
```

Enhancement of the Client II.

- If the computation takes longer than a couple of minutes, checkpointing is also required
- The checkpointing policy is determined by the middleware (BOINC core client), so the `DC_checkClientEvent()` function should be called from time to time to check if a checkpoint is needed

```
DC_ClientEvent *event;  
event = DC_checkClientEvent();  
if (event) switch (event->type) {  
    case DC_CLIENT_CHECKPOINT:  
        do_checkpoint(i); break;  
    ...  
}  
DC_destroyClientEvent(event);
```

Enhancement of the Client II.

- Creating a checkpoint is easy, just write out the loop cycle counter. We have to notify the DC-API of the checkpoint

```
char *file_name;  
FILE *f;
```

```
file_name = DC_resolveFileName(DC_FILE_OUT,  
    DC_CHECKPOINT_FILE);  
f = fopen(file_name, "w");  
fprintf(f, "%d\n", loop_counter);  
fclose(f);  
DC_checkpointMade(file_name);  
free(file_name);
```

Enhancement of the Client II.

- The only thing left is to check for an existing checkpoint at startup and use that as the loop counter's initial value instead of 2 in `is_prime()`

```
file_name = DC_resolveFileName(DC_FILE_IN,  
    DC_CHECKPOINT_FILE);  
if (file_name) { ...  
    fgets(buf, sizeof(buf), f);  
    start = atoi(buf);  
}  
else start = 2;
```

Enhancement of the Master

- The master is a standard UNIX process that will normally be shut down when the project is stopped. So we need checkpointing in the master too
- Since there is no middleware above us, there is no explicit support in DC-API
 - Well, there is for saving the WU states, but we do not need that in this example
- Our case is rather simple: we have just 3 state variables that must be saved (**submitted**, **found** and **errors**)
 - All other state information is kept inside BOINC so we do not have to worry about those
- The master must trap the **SIGINT** and **SIGEXIT** signals and do a checkpoint & exit when one of them is caught

Enhancement of the Master

- Signal catching:

```
#include <signal.h>
```

```
static volatile int exit_req;
```

```
static void signal_handler(int sig) {  
    exit_req = 1;  
}
```

```
main() {  
    struct sigaction sa;  
    memset(&sa, 0, sizeof(sa));  
    sa.sa_handler = signal_handler;  
    sigaction(SIGINT, &sa, NULL);
```

And a Little Help...

- The sources containing all the modifications mentioned in the previous slides are available at:

<http://www.lpds.sztaki.hu/~gombasg/cheat>