Developing a PHP test program for GRID Storage Elements

DESY Summer Student Program 2006
IT Division

Klaus Wiebe
Sep, 11^{th} 2006

1 Introduction

Nowadays it is quite natural that one can access almost infinite amounts of data in the internet from almost everywhere in the world. Fast evolving network and storage technologies ensures scalability and improvement of the required resources. It is not necessary that people know about the complex processes happening when they surf the web, they just know they have to install a browser and purchase some sort of internet access offered by Internet Service Providers (ISPs). What about having access to almost infinite computing resources everywhere in the world? This is where the Grid Computing comes into play. In analogy to the power grid, where people do not know where the electricity has been produced or which way it took, users who want to use the computing grid[1] do not need to know where their data are stored and where their programs are running. They only know that it all happens in the grid. Resource management devices called Resource Brokers (RB) are making these assignment decisions based on free capacities. A software called middleware provides a user interface which doesn't require users to unterstand everything that happens "behind the scenes" (illustrated in Figure 1). The actual idea about the grid was developed by Ian Foster and Carl Kesselmann in 1998 ("The Grid is a way of using the Internet to share and manage computing resources that are distributed around the globe.").
Planning and realization of the currently largest grid infrastructure was initiated by CERN since LHC experiments starting in 2007 will produce large amounts of data (approx. 15 PB per year) that the CERN computing site cannot deal with it on its own. Data not only has to be stored, but also be provided to about 5000 scientists who want to analyze it. CERN's plans were (and still are being) supported by the European Union within the scope of the EGEE[2] program. Scientists developed the LCG (LHC Computing Grid) middleware which provides users with commands to work with the LCG and make the LCG work for them. Choosing grid technology also had other reasons, like secure user authentication, decentralized control and user organization (more about this in the next section).

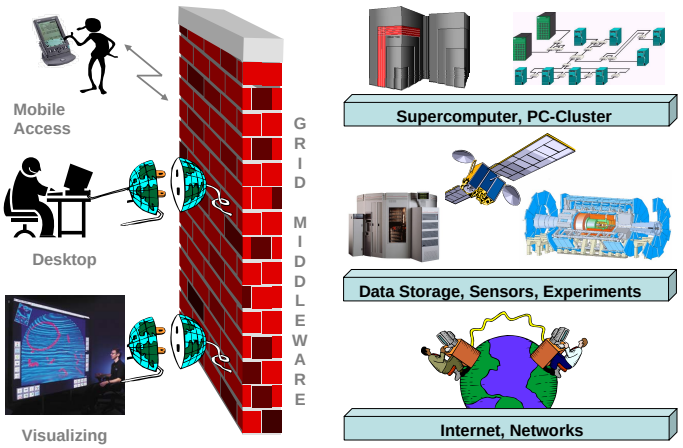


Figure 1: *The grid dream*

[1] from now on the word "grid" is used synonymously for "computing grid"
[2] Enabling Grids for E-sciencE, www.eu-egee.org

Ian Foster ("What is the Grid?", 2002) sums up grid requirements in three major points:
"The grid...

- coordinates resources that are not subject to centralized control - (A Grid integrates and coordinates resources and users that live within different control domains for example, the user's desktop vs. central computing; different administrative units of the same company; or different companies; and addresses the issues of security, policy, payment, membership, and so forth that arise in these settings. Otherwise, we are dealing with a local management system.)
- uses standard, open, general-purpose protocols and interfaces - (A Grid is built from multi-purpose protocols and interfaces that address such fundamental issues as authentication, authorization, resource discovery, and resource access. [...], it is important that these protocols and interfaces be standard and open. Otherwise, we are dealing with an application-specific system.)
- is to deliver nontrivial qualities of service - (A Grid allows its constituent resources to be used in a coordinated fashion to deliver various qualities of service, relating for example to response time, throughput, availability, and security, and/or co-allocation of multiple resource types to meet complex user demands, so that the utility of the combined system is significantly greater than that of the sum of its parts.)"

2 LCG architecture

The LCG which is used for this project is organized in a layer structure called **tier**s. Each grid site which belongs to a certain tier layer has to provide certain services. A tier 1 center has to offer more capacity and stability than a tier 2 center. There are 10^0 (=1) tier-0 centers, approx. 10^1 (=10) tier-1 centers and approx. 10^2 (=100) tier-2 centers. The CERN site is the tier 0 center in LCG, the (only) german tier 1 center is at Forschungszentrum Karlsruhe (FZK), DESY is a tier 2 grid site and will be enolved in LHC experiments ATLAS and CMS, providing all core services. DESY is running grid activities since 2003 and its grid infrastructure is heavily used by HERA and ILC experiments.
Resource management is provided by **Resource Brokers** (RBs) which decide to which site a job (for computing) should be forwarded when a user sends them "to the grid" (see Figure 2). Made decisions should not be the user's concern and are based on free capacities. Alternatively it is always possible to specify to which site a job or file should be sent.
In High Energy Physics data is very precious. In the grid this data is stored on so-called **Storage Elements** (SE) allowing key storage functions like storing, copying, deleting and replicating files to other storage elements. As the same file could have several copies on different SEs there is a file catalog containing one unique logical identifier for a file and linked with the logical identifier the physical location(s). A file is only considered a grid file if it is both registered in the file catalog **and** physically present in a SE.
If computing (e.g. Monte Carlo Simulation) is required, then the desired program (could be a shell script or a C binary) is sent in form of a job to a **Computing Element** (CE). Every grid site has at least one CE. The .jdl[3] job file specifies the contents of the Input Sandbox, the Output Sandbox, the name of the executable file and other parameters. The executable file is the program which is meant to be computed. The Input Sandbox specifies names of files which are required by the executable, these files will also be sent to the Computing Element (together with the .jdl file and the main program). The output sandbox specifies names of files which contain program outputs (such as StdOut and StdErr) and possibly produced files during runtime. Input and Output sandbox should only be used for small files (up to 10 MB), larger files are to be stored on a storage element. If the program requires larger files or produces larger files, these files should be

[3]Job discription language

stored on SEs. When computing is finished the user may transfer the output files to his PC, then the job will get status "cleared".
The processing of a job is divided into several steps which are: submitted, waiting, ready, scheduled, running, done. When a job is done, the user may get his results (OutputSandbox), then the job state will change to "cleared".

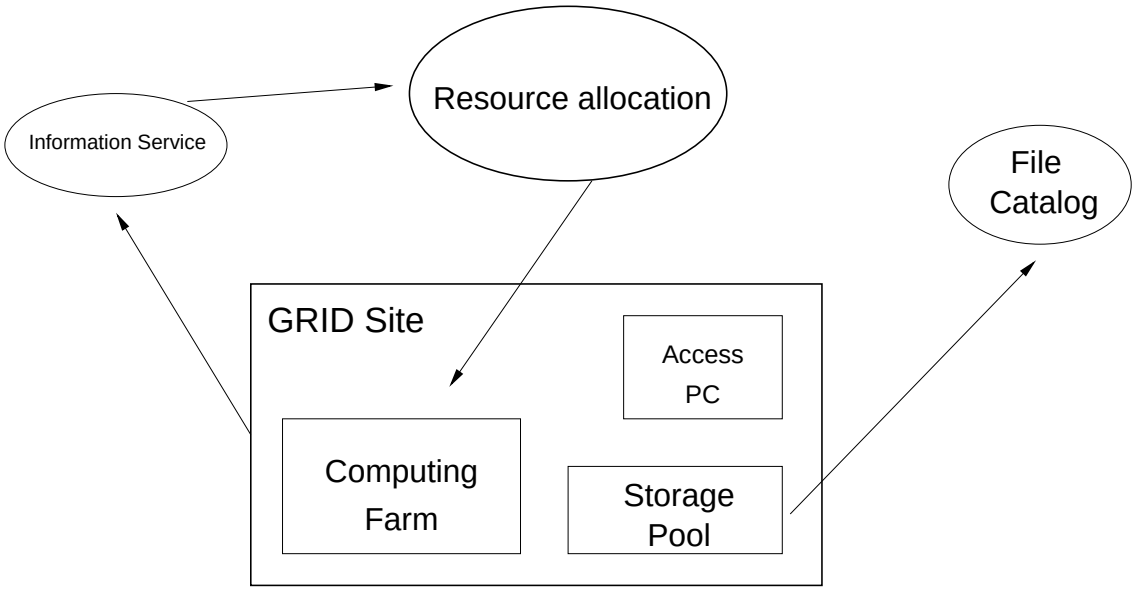


Figure 2: *Abstract component structure in LCG*

Worker Nodes (WN) are computers on a farm behind a CE. The CE manages and delegates jobs to these worker nodes, depending on workload. WNs usually run Scientific Linux, currently there are about 30,000 WNs in the LCG (DESY has more than 400), distributed all over the globe. When a WN calculates a job, it temporarily offers at least 5 GB of local space for the executed program. Hence LCG middleware is always installed on WNs, the executable can access SEs like a user using a user interface.

Resource sharing in the Grid means direct access to computers, software, data, etc. and therefore is controlled. Rules define what is shared and who is allowed to share.
"A set of individuals and/or institutions defined by such sharing rules is what we call a **Virtual Organization** (VO)" (I. Foster, C. Kesselmann, S. Tuecke (2000))
In HEP, typically a research collaboration (like Atlas or Zeus) forms a VO.
To gain access to the grid and become a member of one VO, one must contact a **Certification Authority** (CA) and request a valid X.509 certificate. The role of a CA is to guarantee that a user is who he claims to be and is entitled to own his certificate. In general CAs are organized geographically and by research institutes.
If everything goes fine, one will receive a grid certificate, this would be some files containing the user's public and private key (RSA encryption). When initiating an (encrypted) session, a proxy certificate is created. A proxy certificate is a delegated user credential that authenticates the user in every secure interaction, and has a limited lifetime: in fact, it prevents having to use one's own certificate, which could compromise its safety. By default, created proxy certificates are valid for 12 hours. Altering this span of time is possible, however longer proxy lifetimes imply bigger security risks.

The LCG middleware (or the glite[4] middleware) provides the user with commands to interact with grid resources. Listed below you find some important commands:

- **grid-proxy-init**
  By creating a proxy certificate a secure session is started. Execution of commands which require an authenticated user is possible then.
- **lcg-infosites**
  Displays which SEs or CEs can be used by a specific VO.
- **lfc-ls**
  Allows viewing folder contents of the logical file catalog.
- **lfc-mkdir**
  Makes a directory in the LCG File Catalog (LFC).
- **lcg-cr**
  Creates a file entry in the LFC and physically uploads this file to a SE (SE can be specified).
- **lcg-rep**
  Replicates the file to another SE.
- **lcg-lr**
  Views all replicas to a specific LFC location.
- **lcg-del**
  Deletes a file.
- **edg-job-list-match**
  Displays CEs which could accept a specified job.
- **edg-job-submit**
  Sends jdl job file and Input Sandbox to a CE (CE may be specified).
- **edg-job-status**
  Requests and displays current status of a submitted job.
- **edg-job-get-output**
  Transfers the Output Sandbox to the user interface. Job will have status ”cleared” when successfully getting the output.

All of these commands expect parameters, e.g. the VO or destination grid site. Parameters and explanations can be found in the LCG2 user guide available at cern.ch .

[4]LCG and glite middleware are merged together (new name: glite 3.0)

3 Motivation

Since software for grid components is still being developed, resources are vulnerable to failures and software/hardware needs to be tested well to assure stable operation and an early detection of errors. Multiple parallel requests for larger files (1 - 1000 MB) demanding reliable storage hardware, bandwidth capacity and good software is one possibility of realizing performance tests of Storage Elements.

Manual testing via shell commands is possible, but obviously unefficient and long-winded. Writing shell scripts would avoid entering same commands very often, but testing with this method still needs much time if demands have changed and different tests need to be set up. A comfortable graphical user interface is required which permits choosing from various options, dynamically internally creating shell scripts, executing grid commands, measuring results and presenting the gained data in diagrams.

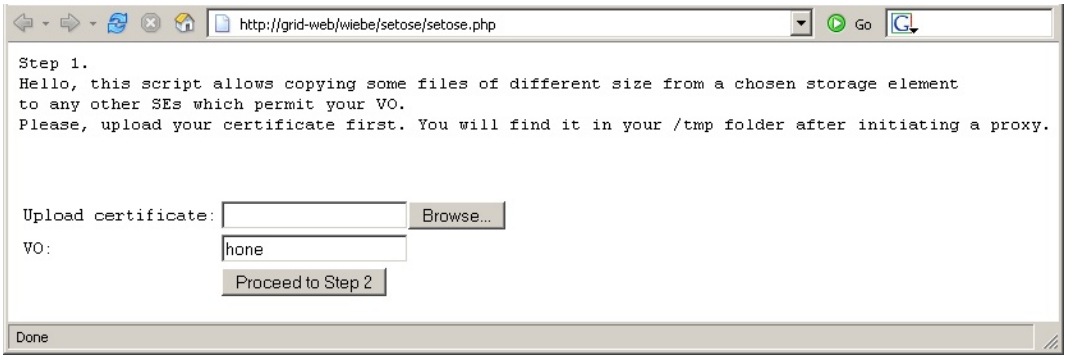
4 The PHP program

PHP[5] was chosen as a powerful and modern server-side scripting language producing HTML output which easily can be accessed by everyone on the DESY campus via web interface (with the browser of your choice).

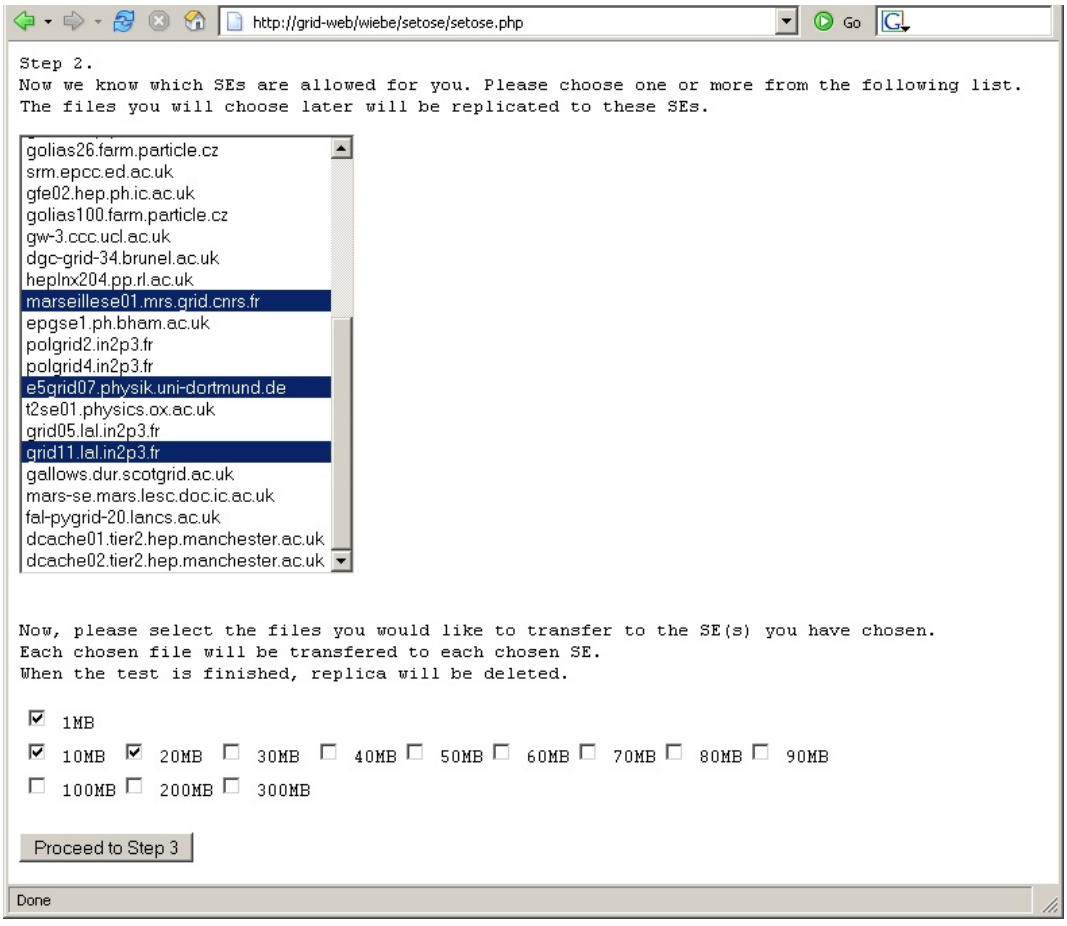
My program tests storage elements by replicating files of different size from the SE which should be tested (in my case this was a DESY SE) to specified other SEs. Transfer time is measured and displayed and the user will also see if the destination SEs are functioning properly. The program is divided into four steps which guide the user through the process. To be more concrete, only the first two steps require user input, the third stage shows the status while replicating and the last one displays the resulting charts with the measured time. Each step features a short explanation what it is about.
However, some preparatory actions need to be performed. Dummy files of different sizes (1; 10; 20; 30; 40; 50; 60; 70; 80; 90; 100; 200 and 300 MB) need to be produced and uploaded to a SE. They shall all be named xx.dat (xx is the filesize, e.g. 20), lie within one folder in the LFC and be physically present on only one SE: The logical path and the name of the SE have to be specified at the beginning of the source code. This short documentation can also be found when viewing the php file's source code.
I will now give a short overview about the several steps and their features:

- **Step 1**
  Step one of the program handles VO specification and upload of the proxy certificate file. As described above the proxy certificate is valid for a specific period of time (called session) after initiation, by default 12 hours.
  As grid commands have to be executed by Apache, it needs a proxy and therefore a certificate. Apache of course can't be given a certificate, because it is not a person. But Apache can "lend" the user's certificate, that's the reason why it has to be uploaded.
  Finally, the user's VO has to be specified, because in most LCG commands the VO must be given (by the parameter `--vo [vo-name]`).

[5] short form for "PHP hypertext preprocessor", www.php.net

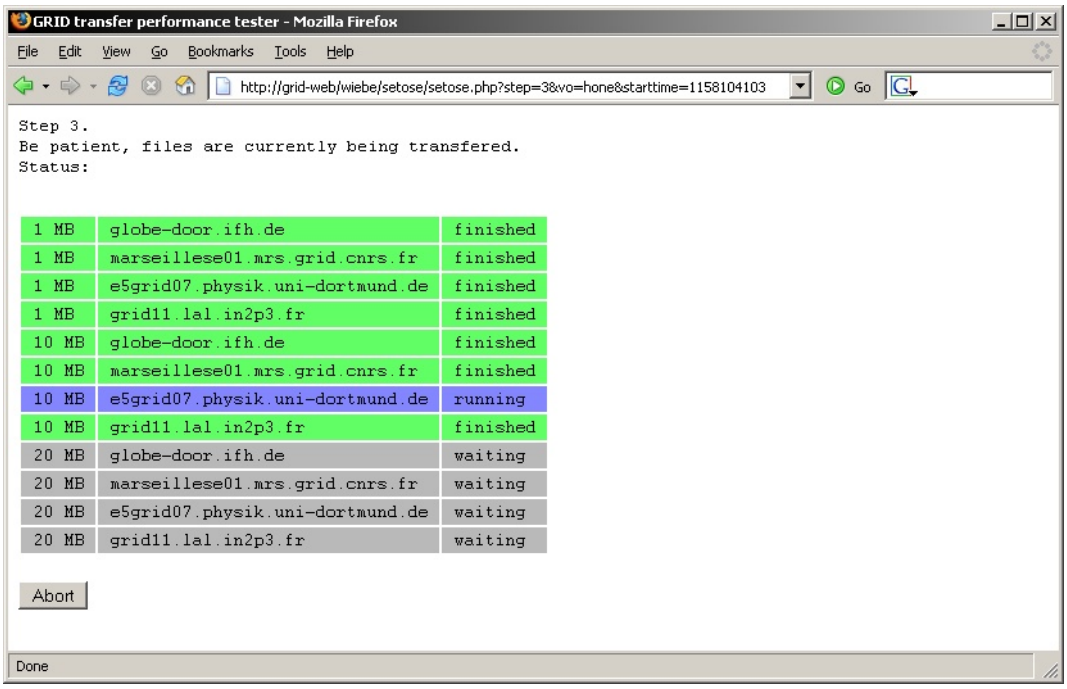


- **Step 2**
  After giving start-up information in step one, the program finds out which SEs are allowed for the VO specified in the first step. It finds out by executing `lcg-infosites --vo [vo] se`. The desired information (the names of the SEs) will be filtered from the command's output and put as options into a select box (see screenshot).
  Additionally the program makes a temp folder for future shell scripts, result files and the configuration file. These will be produced in step 3.



In the second part of this step file sizes have to be chosen. When replicating the files in step 3, each file will be replicated parallely to all chosen SEs, then it's the turn of the next filesize.

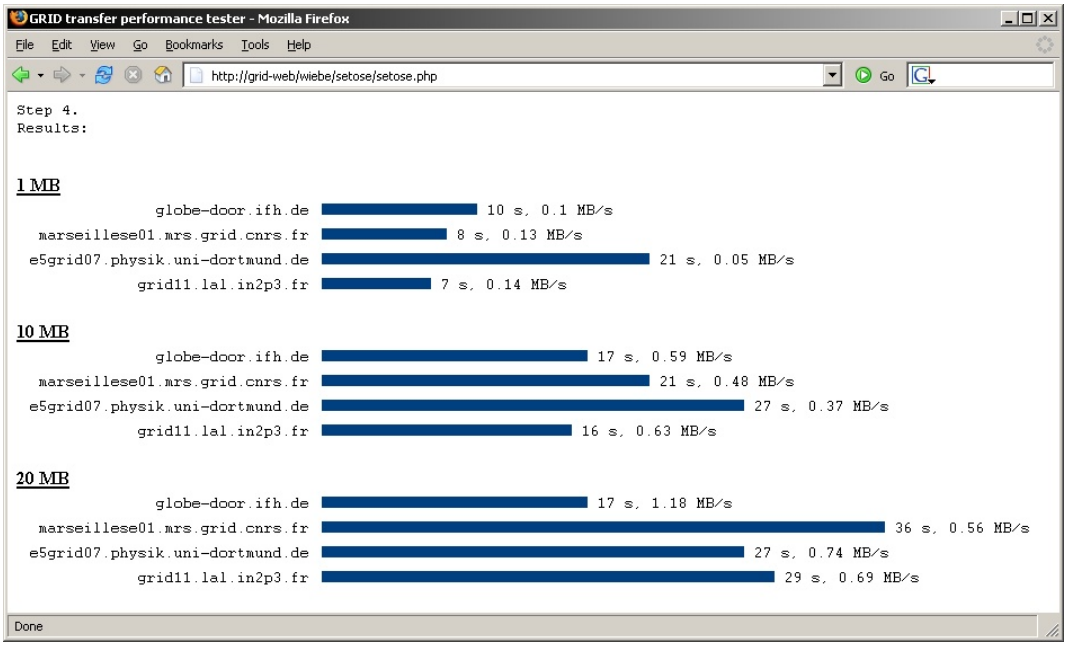
- **Step 3**
  Step 3 is the most important stage during program runtime. First, it creates a configuration file inside the temp folder, containing all chosen filesizes and SEs. Then the shell scripts are created, one for each combination of SE and filesize. The last row (`time lcg-rep --vo [vo] [logicallocation][filesize].dat -d [se]`) initiates the file replication. The error output of this command is lead in a result file (`.res`), however not all shell scripts are executed simultaneously. Replications start in logical units (one filesize at once, to all SEs, then the next filesize to all SEs, ...). By self-refreshing every 5 seconds the program regularly checks progress of all transfers. This is done by looking at the filesize. If a corresponding result file (to a SE filesize combination .sh file) does not exist this means that the transfer has not started yet. If the filesize is zero the transfer has started and is currently running, because no time results have been written yet. In case the result filesize is about 40 bytes the transfer has finished successfully. If it is significantly greater than 40 bytes an error has occured during replication or the destination SE has not granted access. In this case the transfer status will changed to 'failed' and the name of the SE will be deleted from the configuration file. The time from transfer start is always counted by the program itself and if the average transfer rate falls below 0.2 MB/s this transfer will be aborted and the result file's size will be set to 66 bytes. The status of this transfer will then change to 'time out' due to 66 bytes filesize. The name of the SE will be deleted in the configuration file.



There is always the option of aborting all transfers by clicking on the Abort button, because e.g. transfering a 300 MB file at 0.2 MB/s takes over 25 minutes.
In case all .sh files have corresponding .res files and the filesizes of all .res files are greater than zero all replication processes are finished and the program starts deleting all replicas by exucuting the command `lcg-del --vo [vo] [physical_location]` for every physically present file (except the ones on the source SE).
Reached this state the program automatically proceeds to step 4 where results will be displayed.

- **Step 4**

  When examining the configuration file the program knows which results to display. By extracting time information out of the configuration file it can calculate ratios, which yield to graphical time bars (see screenshot). Effective transfer rate (including overhead) is calculated from filesize and measured time.
  Looking at the resulting bars one can notice that measured times do not develop proportionally to the filesizes. This is due to data overhead during replication start. When a transfer starts between two communicating components handshakes and authentication data has to be exchanged.



  Finally, the shell scripts and the proxy certificate file `x509up_u48` in the /tmp path are removed.

Note, that the php program will reject working when there are still .sh files (shell scripts) in the temp folder. This is checked in step 1. In this case the program thinks that another instance of it is running somewhere else (which is likely to be true). However, it could be that the program was terminated ungently the last time it was used. In the usual case the shell scripts are deleted in step 4.

5 Thanks to...

... Andreas Gellrich for providing information and pictures for this report and my talk.

... Christoph Wissing and Andreas Gellrich for supervising, helping, giving interesting information and helpful suggestions.

... Benjamin Eberhardt for helping out with xfig.

... the people giving interesting lectures.

... the IT secretaries for sharing their room.

... Prof. Joachim Meyer, Andrea Schrader and the other summerstudent organization staff for perfectly organizing the stay, the lectures, H1 visitation, etc.