CREATING BOINC PROJECTS

February 04 2007

Distributed computation with BOINC

Developing a BOINC application

Creating a BOINC project

1. Overview of BOINC

BOINC is a software platform for distributed computing using volunteered computer resources. The BOINC's features fall into several areas:

Resource sharing among independent projects

Many different projects can use BOINC. Projects are independent; each one operates its own servers and databases. Participants can participate in multiple projects; they control which projects they participate in, and how their resources are divided among these projects. When a project is down or has no work, the resources of its participants are divided among other projects.

Project features

BOINC provides features that simplify the creation and operation of distributed computing projects.

- **Flexible application framework**
  Existing applications in common languages (C, C++, Fortran) can run as BOINC applications with little or no modification. An application can consist of several files (e.g. multiple programs and a coordinating script). New versions of applications can be deployed with no participant involvement.
- **Security**
  BOINC protects against several types of attacks. For example, digital signatures based on public-key encryption protect against the distribution of viruses.
- **Multiple servers and fault-tolerance**
  Projects can have separate scheduling and data servers, with multiple servers of each type. Clients automatically try alternate servers; if all servers are down, clients do exponential backoff to avoid flooding the servers when they come back up.
- **Source code availability**
  BOINC is distributed under the Lesser GNU Public License. However, BOINC applications need not be open source.
- **Support for large data**
  BOINC supports applications that produce or consume large amounts of data, or that use large amounts of memory. Data distribution and collection can be spread across many servers, and participant hosts transfer large data unobtrusively. Users can specify limits on disk usage and network bandwidth. Work is dispatched only to hosts able to handle it.

Participant features

BOINC provides the following features to participants:

- **Multiple participant platforms**
  The BOINC core client is available for most common platforms (Mac OS X, Windows, Linux and other Unix systems). The client can use multiple CPUs.
- **Web-based participant interfaces**
  BOINC provides web-based interfaces for account creation, preference editing, and participant status display. A participant's preferences are automatically propagated to all their hosts, making it easy to manage large numbers of hosts.
- **Configurable host work caching**
  The core client downloads enough work to keep its host busy for a user-specifiable amount of time. This can be used to decrease the frequency of connections or to allow the host to keep working during project downtime.

2. Which applications are suitable for BOINC?

BOINC is designed to support applications that have large computation requirements, storage requirements, or both. A project can gain access to many TeraFLOPs of CPU power and Terabytes of storage.

However, because the resources of a BOINC project are volunteered, (and therefore unreliable and sporadically-connected) an application must have several properties to effectively use BOINC:

Public appeal
: An application must be viewed as interesting and worthwhile by the public in order to gain large numbers of participants. A project must have the resources and commitment to maintain this interest, typically by creating a compelling web site and by generating interesting graphics in the application.

Independent parallelism
: The application must be divisible into parallel parts with few or no data dependencies.

Low data/compute ratio
: Input and output data are sent through commercial Internet connections, which may be expensive and/or slow. As a rule of thumb, if your application produces or consumes more than a gigabyte of data per day of CPU time, then it may be cheaper to use in-house cluster computing rather than volunteer computing.

Fault tolerance
: A result returned from a volunteered computer cannot be assumed to be correct. Redundant computing can be used to reduce the error probability, but not all the way to zero. If your application relies on 100% correctness, this may be a problem.

3. Projects and applications

A **project** is a group of one or more distributed applications, run by a single organization, that use BOINC. Projects are independent; each one has its own applications, databases and servers, and is not affected by the status of other projects.

Each is identified by a master URL, which refers to a web page describing the project.

Creating projects is relatively easy. An organization can create projects to do Alpha and Beta testing of applications. Testers can register for these projects, in addition to or instead of the organization's public project.

The server side of a project consists of two parts:

- A **project back end** that supplies applications and work units, and that handles the computational results.
- A **BOINC server complex** that manages data distribution and collection.

The BOINC server complex includes the following components:

- One or more **scheduling servers** that communicate with participant hosts.
- A relational database that stores information about work, results, and participants.
- Utility programs and libraries that allow the project back end to interact with the server complex.
- Web interfaces for participants and developers.
- **Data servers** that distribute input files and collect output files. These are HTTP servers able to handle CGI programs with POST commands. These servers need not be owned or operated by the project. A project might, for example, recruit other organizations to donate network bandwidth by hosting data servers; data could be moved on tape between the project back end and the data servers.

4. Storage

Files and data servers

The BOINC storage model is based on **files**. Examples of files:

- The inputs and/or outputs of computation;
- Components of application: executables, libraries, etc.
- Data for its own sake, e.g. to implement a distributed storage system.

The BOINC core client transfers files to and from project-operated **data servers** using HTTP.

Once a file is created (on a data server or a participant host) it is **immutable**. This means that all replicas of that file are assumed (and required) to be identical. If a file is changed, even by a single byte, it becomes a new file, and must be given a different name than the original.

File properties

Files have various properties, including:

- **Name**: unique identifier for the file.
- **Sticky**: don't delete file on client (see below).
- **Report on RPC**: include a description of this file in scheduler requests.
- **Maximum size**: if an output file exceeds its maximum size, the computation is aborted.

File properties are specified in XML elements that appear, for example, in workunit templates.

File references

In addition to the properties of a file itself, there are several properties of the way a particular application uses a file; this is called a **file reference**.

- **open_name** (or 'logical name'): the name by which the program will refer to the file (BOINC provides a mechanism for mapping logical names to physical names, so that your applications don't have to know the physical names of their input and output files).
- **copy file**: use this if your application doesn't use BOINC's filename-mapping mechanism.
- **optional**: Use this for output files, to indicate that the application doesn't always generate the file, and that its absence should not be treated as an error.

File reference properties are specified in XML elements.

File management

BOINC's default behavior is to delete files when they aren't needed any more. Specifically:

- On the client, input files are deleted when no workunit refers to them, and output files are deleted when no result refers to them. Application-version files are deleted when they are referenced only from superceded application versions.
- On the client, the 'sticky' flag overrides the above mechanisms and suppresses the deletion of the file. The file may deleted by an explicit server request. The file may also be deleted at any time by the core client in order to honor limits on disk-space usage.
- On the server, the file deleter daemon deletes input and output files that are no longer needed. This can be suppressed using the 'no_delete' flag, or using command-line options to the file deleter.

Compression of input files

Starting with version 5.4, the BOINC client is able to handle HTTP Content-Encoding types 'deflate' (zlib algorithm) and 'gzip' (gzip algorithm). The client decompresses these files 'on the fly' and stores them on disk in uncompressed form.

You can use this in two ways:

- Use the Apache 2.0 mod_deflate module to automatically compress files on the fly. This method will work with all BOINC clients, but it will do compression only for 5.4+ clients.
- Compress files and give them a filename suffix such as '.gz'. The name used in your `<file_info>` elements, however, is the original filename without '.gz'.

  Include the following line in httpd.conf:

  ```
  AddEncoding x-gzip .gz
  ```

  This will add the content encoding to the header so that the client will decompress the file automatically. This method has the advantage of reducing server disk usage and server CPU load, but it will only work with 5.4+ clients. Use the 'min_core_version' field of the app_version table to enforce this. You can use this in conjunction because the mod_deflate module allows you to exempt certain filetypes from on-the-fly compression.

Both methods store files uncompressed on the client. If you need compression on the client, you must do it at the application level. The BOINC source distribution includes a version of the zip library designed for use by BOINC applications on any platform.

Compression of output files

If you include the `<gzip_when_done>` tag in an output file description, the file will be gzip-compressed after it has been generated.

5. Platforms

The computers available to a volunteer computing project have a wide range of operating systems and hardware architectures. For example, they may run many versions of Windows (95, 98, ME, 2000, XP) on many processors variants (486, Pentium, AMD). Hosts may have multiple processors and/or graphics coprocessors.

A **platform** is a compilation target - typically a combination of a CPU architecture and an operating system. The BOINC database of each project includes a set of platforms. Each platform has a **name** and a **description** of the range of computers it can handle. Each application version is associated with a particular platform.

For coherence between projects, you should use only the following platforms: If you want to add a different platform, please contact us.

name	description
windows_intelx86	Microsoft Windows (98 or later) running on an Intel x86-compatible CPU
i686-pc-linux-gnu	Linux running on an Intel x86-compatible CPU
x86_64-pc-linux-gnu	Linux running on an AMD x86_64 or Intel EM64T CPU
powerpc-apple-darwin	Mac OS X 10.3 or later running on Motorola PowerPC
i686-apple-darwin	Mac OS 10.4 or later running on Intel
sparc-sun-solaris2.7	Solaris 2.7 running on a SPARC-compatible CPU
sparc-sun-solaris	Solaris 2.8 or later running on a SPARC-compatible CPU
sparc64-sun-solaris	Solaris 2.8 or later running on a SPARC 64-bit CPU

A platform name is compiled into the BOINC client. The client reports its platform to the scheduling server, and the scheduling server sends work to a host only if there is an application version for the same platform.

In some cases, you may want to associate the a single executable with multiple platforms. For example, a Mac/Intel host is able to run Mac/PPC applications in emulation mode. If you are unable to compile your application for Mac/Intel, you can take your Mac/PPC binary and add it as a Mac/Intel app version; this will allow Mac/Intel hosts to participate in your project.

Application optimization for specific architectures

BOINC allows applications to exploit specific architectures, but places the burden of recognizing the architecture on the application. In other words, if you want to make a version of your application that can use the AMD 3DNow instruction set, don't create a new **windows_amd_3dnow** platform. Instead, make a version for the **windows_intelx86** platform that recognizes when it's running on a 3DNow machine, and branches to the appropriate code.

This excludes the combinatorial explosion of versions and architectures from the internals of BOINC.

Web-site statistics breakdown by architecture

BOINC collects architecture details about each completed result to allow detailed statistical breakdowns.

First, the core client attempts to find the CPU vendor, the CPU model, the OS name, and the OS version. These are stored in the host record.

Second, applications that recognize even more specific architecture information can pass it back to the core client using the **boinc_architecture()** function from the BOINC API. This passes a string (project-specific, but typically in XML) to the core client, which records it in the **architecture_xml** field of the **result** database record. For example, the application might pass a description like

```
<has_3dnow_instructions/>
<graphics_board>ATI Rage 64MB</graphics_board>
```

This makes it possible, for example, to report average or total performance statistics for 3DNow hosts contrasted with other Intel-compatible hosts.

Tools

Platforms are maintained in the **platform** table in the BOINC DB, and can be created using the xadd utility.

6. Applications and versions

An **application** consists of a program (perhaps with versions for different platforms) and a set of workunits and results. A project can operate many applications. Applications are maintained in the **application** table in the BOINC DB, and can be created using the xadd utility.

An application program may go through a sequence of versions. A particular version, compiled for a particular platform, is called an **application version**. An application version can consist of multiple files: for example, a controller script, pre- and post-processing programs, and a primary program.

Each application version has an integer version number. Version numbers should be used consistently across platforms; Windows version 304 should be computationally identical to Mac version 304.

Each application has a **minimum version**. When a client is sent work for an application, it is also sent the latest application version for its platform. It is sent work only if this version is the minimum or greater.

Application versions can be created using update_versions. Descriptions of application versions are stored in the **app_version** table in the BOINC DB.

7. Workunits

A **workunit** describes a computation to be performed. It is represented by a row in the 'workunit' database table. BOINC provides a utility program and C function for creating workunits.

A workunit has several attributes that are specified when the workunit is created:

name	A text string, unique across all workunits in the project.
application	Which application will perform the computation. A workunit is associated with an application, not with a particular version or range of versions. If the input data format changes in a way that is incompatible with older versions, you must create a new application. This can often be avoided by using XML data format.

input files A list of the input files: their names, and the names by which the application refers to them. Typically these file are downloaded from a data server. However, if the <generate_locally/> element is present, the file is generated on the client (typically by an earlier instance of the same application). Applications should use file locking to prevent two jobs from generating the file at the same time.

priority Higher-priority work is dispatched first

batch An integer; can be used to operate (cancel, change priority etc.) on groups of workunits.

Resource estimates and bounds

rsc_fpops_est An estimate of the average number of floating-point operations required to complete the computation. This used to estimate how long the computation will take on a given host.

rsc_fpops_bound A bound on the number of floating-point operations required to complete the computation. If this bound is exceeded, the application will be aborted.

rsc_memory_bound An estimate of application's largest working set size. The workunit will only be sent to hosts with at least this much available RAM.

rsc_disk_bound A bound on the maximum disk space used by the application, including all input, temporary, and output files. The workunit will only be sent to hosts with at least this much available disk space. If this bound is exceeded, the application will be aborted.

Redundancy and scheduling attributes

delay_bound An upper bound on the time (in seconds) between sending a result to a client and receiving a reply. The scheduler won't issue a result if the estimated completion time exceeds this. If the client doesn't respond within this interval, the server 'gives up' on the result and generates a new result, to be assigned to another client.

Set this to several times the average execution time of a workunit on a typical PC. If you set it too low, BOINC may not be able to send some results, and the corresponding workunit will be flagged with an error. If you set it too high, there may a corresponding delay in getting results back.

min_quorum The minimum size of a 'quorum'. The validator is run when there are this many successful results. If a strict majority agree, they are considered correct. Set this to two or more if you want redundant computing.

target_nresults How many results to create initially. This must be at least **min_quorum**. It may be more, to reflect the ratio of result loss, or to get a quorum more quickly.

max_error_results If the number of client error results exceeds this, the work unit is declared to have an error; no further results are issued, and the assimilator is triggered. This safeguards against workunits that cause the application to crash.

max_total_results If the total number of results for this workunit exceeds this, the workunit is declared to be in error. This safeguards against workunits that are never reported (e.g. because they crash the core client).

max_success_results If the number of success results for this workunit exceeds this, and a consensus has not been reached, the workunit is declared to be in error. This safeguards against workunits that produce nondeterministic results.

A workunit has a dynamic attribute:

error mask A bit mask of various error conditions:

- **WU_ERROR_COULDNT_SEND_RESULT**: The BOINC scheduler was unable to send the workunit to a large number (~100) of hosts, probably because its resource requirements (disk, memory, CPU) were too large for the hosts, or because no application version was available for the hosts' platforms. In this case BOINC 'gives up' on the workunit.
- **WU_ERROR_TOO_MANY_ERROR_RESULTS**: Too many results with error conditions (upload/download problem, client crashes) have been returned for this work unit.
- **WU_ERROR_TOO_MANY_SUCCESS_RESULTS**: Too many successful results have been returned without consensus. This indicates that the application may be nondeterministic.
- **WU_ERROR_TOO_MANY_TOTAL_RESULTS**: Too many total results have been sent for this workunit.

If any of these conditions holds, BOINC 'gives up' on the workunit and doesn't dispatch more results for it.

8. Results

A **result** describes an instance of a computation, either unstarted, in progress, or completed. Each result is described by an entry in the 'result' database table. A project does not explictly create results; rather, BOINC creates them automatically for workunits, based on the redundancy parameters of the workunit.

The static attributes of a result include:

name	A text string, unique across all results in the project.
workunit name	Identifies the associated workunit
output files	A list of the names of the output files, and the names by which the application refers to them.

A result has a dynamic attribute:

server state	Values include: • Inactive (not ready to dispatch) • Unsent (ready to sent to a client, but not sent) • In progress (sent, not done) • Done successfully • Timed out • Done with error • Not needed (work unit was finalized before this result was sent)

Additional attributes are defined after the result is completed:

host	The host that executed the computation
exit status	The exit status (0 if success)
CPU time	The CPU time that was used.
output file info	The sizes and checksums of its output files
stderr	The stderr output of the computation
received time	The time when the result was received.

9. Redundancy and errors

A BOINC 'result' abstracts an instance of a computation, possibly not performed yet. Typically, a BOINC server sends 'results' to clients, and the clients perform the computation and replies to the server. But many things can happen to a result:

- The client computes the result correctly and returns it.
- The client computes the result incorrectly and returns it.
- The client fails to download or upload files.
- The application crashes on the client.
- The client never returns anything because it breaks or stops running BOINC.
- The scheduler isn't able to send the result because it requires more resources than any client has.

BOINC provides a form of redundant computing in which each computation is performed on multiple clients, the results are compared, and are accepted only when a 'consensus' is reached. In some cases new results must be created and sent.

BOINC manages most of the details; however, there are two places where the application developer gets involved:

- **Validation:** This performs two functions. First, when a sufficient number (a 'quorum') of successful results have been returned, it compares them and sees if there is a 'consensus'. The method of comparing results (which may need to take into account platform-varying floating point arithmetic) and the policy for determining consensus (e.g., best two out of three) are supplied by the application. If a consensus is reached, a particular result is designated as the 'canonical' result. Second, if a result arrives after a consensus has already been reached, the new result is compared with the canonical result; this determines whether the user gets credit.
- **Assimilation:** This is the mechanism by which the project is notified of the completion (success or unsuccessful) of a work unit. It is performed exactly once per work unit. If the work unit was completed successfully (i.e. if there is a canonical result) the project-supplied function reads the output file(s) and handles the information, e.g. by recording it in a database. If the workunit failed, the function might write an entry in a log, send an email, etc.

In the following example, the project creates a workunit with
min_quorum = 2
target_nresults = 3
max_delay = 10

BOINC automatically creates three results, which are sent at various times. At time 8, two successful results have returned so the validator is invoked. It finds a consensus, so the work unit is assimilated. At time 10 result 3 arrives; validation is performed again, this time to check whether result 3 gets credit.

```
time        0   1   2   3   4   5   6   7   8   9   10  11  12  13  14

            created                         validate; assimilate
WU          x                               x  x
                created sent            success
result 1        x       x---------------x
                created sent                success
result 2        x       x-------------------x
                created     sent                    success
result 3        x           x-----------------------x
```

In the next example, result 2 is lost (i.e., there's no reply to the BOINC scheduler). When result 3 arrives a consensus is found and the work unit is assimilated. At time 13 the scheduler 'gives up' on result 2 (this allows it to delete the canonical result's output files, which are needed to validate late-arriving results).

```
time        0   1   2   3   4   5   6   7   8   9   10  11  12  13  14

            created                                 validate; assimilate
WU          x                                       x  x
                created sent            success
result 1        x       x---------------x
```

10. Dealing with numerical discrepancies

Most numerical applications produce different outcomes for a given workunit depending on the machine architecture, operating system, compiler, and compiler flags. For some applications these discrepancies produce only small differences in the final output, and results can be validated using a 'fuzzy comparison' function that allows for deviations of a few percent.

Other applications are 'divergent' in the sense that small numerical differences lead to unpredictably large differences in the final output. For such applications it may be difficult to distinguish between results that are correct but differ because of numerical discrepancies, and results that are erroneous. The 'fuzzy comparison' approach does not work for such applications.

Eliminating discrepancies

One approach is to eliminate numerical discrepancies. Some notes on how to do this for Fortran programs are given in a paper, Massive Tracking on Heterogeneous Platforms and in an earlier text document, both courtesy of Eric McIntosh,

Homogeneous redundancy

BOINC provides a feature called **homogeneous redundancy** to handle divergent applications. You can enable it for a project by including the line

```
<homogeneous_redundancy/>
```

in the config.xml file.

Alternatively, you can enable it selectively for a single application by setting the `homogeneous_redundancy` field in its database record.

When this feature is enabled, the BOINC scheduler will send results for a given workunit only to hosts with the same operation system name and CPU vendor (i.e., the os_name and p_vendor fields of the host description). For example: if a result has been sent to a host of type (Windows XP, Intel), then other results of that workunit will only be sent to hosts of type (Windows XP, Intel).

If homogeneous redundancy is enabled, it may be possible to use strict equality to compare redundant results.

11. Work distribution

A host asks a project for work by including a

```
<work_req_seconds>X</work_req_seconds>
```

element in a scheduler RPC request message. This asks the scheduler to return enough work to keep all the host's processors busy for X seconds, given the host's typical usage (i.e. the fraction of time it's turned off or BOINC is suspended, and the other processes that it executes).

BOINC's work distribution policy addresses the (sometimes conflicting) goals of keeping hosts as busy as possible, while minimizing the impact of

- Hosts that repeatedly return results with error outcomes, due to a host-specific problem.
- Malicious participants who attempt to obtain multiple results of the same workunit, in an attempt to obtain unearned credit or have erroneous results accepted as correct.

Work distribution is constrained by a number of rules:

- A result is sent only if an application version is available for the host's platform. If the application's min_version field is nonzero, the version number must at list this value.
- A result is not sent if its disk or memory requirements are not met by the host.
- A result is not sent if the estimated latency (based on the host's CPU speed and usage parameters) exceeds the workunit's latency bound.
- A result is not sent if the project has specified a one result per user per workunit flag, and a result of the same workunit has already been sent to a host belonging to the same user.
- A result is not sent if the project has specified a daily result quota per host, and the host has already been sent that many results.
- A project can specify a limit on the number of results sent per RPC.
- A result is not sent if homogeneous redundancy is enabled and another result of the same workunit has been sent to a different type of host.

In general, the BOINC scheduler responds to a work request by enumerating unsent results from the database, filtering them by the above criteria, sending them to the host, and continuing until requested duration X is reached.

For projects that have very large input files, each of which is used by many workunit, BOINC offers an alternative work distribution policy called locality scheduling.

12. Locality scheduling

Locality scheduling is intended for projects for which

- Each workunit has a large input file (it may have other smaller input files as well).
- Each large input file is used by many workunits.

The goal of locality scheduling is to minimize the amount of data transfer to hosts. In sending work to at given host, the scheduler tries to send results that use input files already on the host.

To use locality scheduling, projects must do the following:

- Workunit names must be of the form FILENAME__*, where FILENAME is the name of the large input file used by that workunit. These filenames cannot contain '__'.
- The <file_info> for each large input file must contain the tags

```
<sticky/>
<report_on_rpc/>
```

- The config.xml file must contain

```
<locality_scheduling/>
```

Locality scheduling works as follows:

- Each scheduler RPC contains a list of the large files already on the host, if any.
- The scheduler attempts to send results that use a file already on the host.
- For each file that is on the host and for which no results are available for sending, the scheduler instructs the host to delete the file.

On-demand work generation

This mechanism, which is used in conjunction with locality scheduling, lets a project create work in response to scheduler requests rather than creating all work ahead of time. The mechanism is controlled by an element in config.xml of the form:

```
<locality_scheduling_wait_period> N </locality_scheduling_wait_period>
```

where N is some number of seconds.

When a host storing file X requests work, and there are no available results using X, then the scheduler

touches a 'trigger file'

```
PROJECT_ROOT/locality_scheduling/need_work/X
```

The scheduler then sleeps for N seconds, and makes one additional attempt to find suitable unsent results.

The project must supply a 'on-demand work generator' daemon program that scans the need_work directory. If it finds an entry, it creates additional workunits for the file, and the transitioner then generates results for these workunits. N should be chosen large enough so that both tasks complete within N seconds most of the time (10 seconds is a good estimate).

The work generator should delete the trigger file after creating work.

In addition, if the work generator (or some other project daemon) determines that no further workunits can be made for a file X, then it can touch a trigger file

```
PROJECT_ROOT/locality_scheduling/no_work_available/X
```

If the scheduler finds this trigger file then it assumes that the project cannot create additional work for this data file and skips the 'notify, sleep, query again' sequence above. Of course it still does the initial query, so if the transitioner has made some new results for an existing (old) WU, they will get picked up.

Implementation notes

Work is organized in a hierarchy:

```
File -> workunit -> result
```

Let's say there are N active hosts and target_nresults=M. Optimally, we'd like to send each file to M hosts, and have them process all the results for that file.

If the one_result_per_user_per_wu rule is in effect, a file may have work but be 'excluded' for a particular user.

Assigning work to a host with no files:

- maintain a working set of N/M files
- when a host with no file requests work, choose a file F uniformly (randomly or sequentially) from the working set.
- if F is excluded for this user, choose a file using a deterministic algorithm that doesn't involve the working set (don't want to do this in general to avoid flocking)

The working set is represented by a directory

```
PROJECT/locality_scheduling/file_working_set/
```

whose contents are names of files in the working set. A project-specific 'working set manager' daemon is responsible for maintaining this.

If the scheduler finds that there are no sendable results for a file, it makes a file with that name in

```
PROJECT/sched_locality/files_no_work/
```

The working set manager should poll this directory and remove those files from the working set. NOTE: BOINC may later create more results for the file, so it may be necessary to add it to the working set again.

Assigning work to a host with a file F:

- send more results for file F. To do this efficiently, we maintain the following invariant: For a given user/file pair, results are sent in increasing ID order.

Some projects may want to generate work incrementally. They can do this by supplying a 'work generator' daemon that polls the directory

```
PROJECT/locality_scheduling/need_work/
```

and creates work for any filenames found there. To enable this, add the element to config.xml; this tells the scheduler how long to wait for work to appear.

NOTE: we assume that all results have app_versions for the same set of platforms. So if any result is rejected for this reason, we give up immediately instead of scanning everything.

13. Trickle messages

Trickle messages let applications communicate with the server during the execution of a workunit. They are intended for applications that have long workunits (multiple days).

Trickle messages may from client to server or vice versa. Messages are XML documents.

Trickle-up messages

Trickle-up messages go from application to server. They are handled by **trickle handler daemons** running on the server. Each message is tagged with a 'variety' (a character string). Each daemon handles messages of a particular variety. (This is used, typically, to distinguish different applications.) Example uses:

- The application sends a trickle-up message containing its current CPU usage, so that users can be granted incremental credit (rather than waiting until the end of the work unit).
- The application sends a trickle-up message containing a summary of the computational state, so that server logic can decide if the computation should be aborted.

To create a trickle handler daemon, modify the program `sched/trickle_handler.C`, replacing the function `handle_trickle()` with your own function. Add an entry in your config.xml to run this program as a daemon.

Trickle-down messages

Trickle-down messages go from server to application. Each one is addressed to a particular host, and must include an element `<result_name>` identifying the result to which the message is addressed. If that result is still running on the host, it is delivered to it. Example uses:

- The server sends a message telling the application to abort.
- The server sends a message containing the user's current total credit.

Trickle messages are asynchronous, ordered, and reliable. Trickle messages are conveyed in scheduler RPC messages, so they may not be delivered immediately after being generated.

14. Security issues in volunteer computing

Many types of attacks are possible in volunteer computing.

- **Result falsification**. Attackers return incorrect results.
- **Credit falsification**. Attackers return results claiming more CPU time than was actually used.
- **Malicious executable distribution**. Attackers break into a BOINC server and, by modifying the database and files, attempt to distribute their own executable (e.g. a virus program) disguised as a BOINC application.
- **Overrun of data server**. Attackers repeatedly send large files to BOINC data servers, filling up their disks and rendering them unusable.
- **Theft of participant account information by server attack**. Attackers break into a BOINC server and steal email addresses and other account information.
- **Theft of participant account information by network attack**. Attackers exploit the BOINC network protocols to steal account information.
- **Theft of project files**. Attackers steal input and/or output files.
- **Intentional abuse of participant hosts by projects**. A project intentionally releases an application that abuses participant hosts, e.g. by stealing sensitive information stored in files.

- **Accidental abuse of participant hosts by projects**. A project releases an application that unintentionally abuses participant hosts, e.g. deleting files or causing crashes.

BOINC provides mechanisms to reduce the likelihood of some of these attacks.

Result falsification

This can be probabilistically detected using redundant computing and result verification: if a majority of results agree (according to an application-specific comparison) then they are classified as correct.

Credit falsification

This can be probabilistically detected using redundant computing and credit verification: each participant is given the minimum credit from among the correct results (or some other algorithm, such as the mean or median of claimed credits).

Malicious executable distribution

BOINC uses code signing to prevent this. Each project has a key pair for code signing. The private key should be kept on a network-isolated machine used for generating digital signatures for executables. The public key is distributed to, and stored on, clients. All files associated with application versions are sent with digital signatures using this key pair.

Even if attackers break into a project's BOINC servers, they will not be able to cause clients to accept a false code file.

BOINC provides a mechanism by which projects can periodically change their code-signing key pair. The project generates a new key pair, then (using the code-signing machine) generates a signature for the new public key, signed with the old private key. The core client will accept a new key only if it's signed with the old key. This mechanism is designed to prevent attackers from breaking into a BOINC server and distributing a false key.

Denial of server attacks on data servers

Each result file has an associated maximum size. Each project has a **upload authentication key pair**. The public key is stored on the project's data servers. Result file descriptions are sent to clients with a digital signature, which is forwarded to the data server when the file is uploaded. The data server verifies the file description, and ensures that the amount of data uploaded does not exceed the maximum size.

Theft of participant account information by server attack

Each project must address theft of private account information (e.g. email addresses) using conventional security practices. All server machines should be protected by a firewall, and should have all unused network services disabled. Access to these machines should be done only with encrypted protocols like SSH. The machines should be subjected to regular security audits.

Projects should be undertaken only the organizations that have sufficient expertise and resources to secure their servers. A successful attack could discredit all BOINC-based projects, and public-participation computing in general.

Theft of participant account information by network attack

Attackers sniffing network traffic could get user's account keys, and use them to get the user's email address, or change the user's preferences. BOINC does nothing to prevent this.

Theft of project files

The input and output files used by BOINC applications are not encrypted. Applications can do this themselves, but it has little effect since data resides in cleartext in memory, where it is easy to access with a debugger.

Intentional abuse of participant hosts by projects

BOINC does nothing to prevent this (e.g. there is no 'sandboxing' of applications). Participants must understand that when they join a BOINC project, they are entrusting the security of their systems to that project.

Accidental abuse of participant hosts by projects

BOINC prevents some problems: for example, it detects when applications use too much disk space, memory, or CPU time, and aborts them. But applications are not 'sandboxed', so many types of accidental abuse are possible. Projects can minimize the likelihood by pre-released application testing. Projects should test their applications thoroughly on all platforms and with all input data scenarios before promoting them to production status.

15. The BOINC application programming interface (API)

The BOINC API is a set of C++ functions. Most of the functions have a C interface, so that they can be used from programs written in C and other languages. Unless otherwise specified, the functions return an integer error code; zero indicates success.

BOINC applications may generate graphics, allowing them to provide a screensaver. This API is described here.

BOINC applications may consist of several programs that are executed in sequence; these are called **compound applications**. This API is described here.

Initialization and termination

Applications should initialize diagnostics before any other BOINC calls.

Initialization for graphical and compound applications are described elsewhere (see links above). Other applications must call

```
int boinc_init();
```

before calling other BOINC functions or doing I/O.

When the application has completed it must call

```
int boinc_finish(int status);
```

`status` is nonzero if an error was encountered. This call does not return.

Is your application running under the control of the BOINC client?

BOINC applications can be run in "standalone" mode for testing, or under the control of the BOINC client. You might want your application to behave differently in the two cases. For example you might want to output debugging information if the application is running standalone. To determine if the application is running in standalone mode or under the control of the BOINC client, call

```
int boinc_is_standalone(void);
```

This returns non-zero (True) if the application is running standalone, and zero (False) if the application is running under the control of the BOINC client.

Resolving file names

Applications that use named input or output files must call

```
int boinc_resolve_filename(char *logical_name, char *physical_name, int len);
```

or

```
int boinc_resolve_filename_s(char *logical_name, std::string& physical_name);
```

to convert logical file names to physical names. For example, instead of

```
f = fopen("my_file", "r");
```

the application might use

```
string resolved_name;
retval = boinc_resolve_filename("my_file", resolved_name);
if (retval) fail("can't resolve filename");
f = fopen(resolved_name.c_str(), "r");
```

`boinc_resolve_filename()` doesn't need to be used for temporary files.

I/O wrappers

16. The BOINC diagnostics API

BOINC applications can call

```
int boinc_init_diagnostics(int flags)
```

to initialize various diagnostic functions. **This call should be made early in the program - before boinc_init() - so that error info is routed appopriately.** `flags` is formed by or'ing together a subset of the following flags:

```
#define BOINC_DIAG_DUMPCALLSTACKENABLED     0x00000001L
#define BOINC_DIAG_HEAPCHECKENABLED         0x00000002L
#define BOINC_DIAG_MEMORYLEAKCHECKENABLED   0x00000004L
#define BOINC_DIAG_ARCHIVESTDERR            0x00000008L
#define BOINC_DIAG_ARCHIVESTDOUT            0x00000010L
#define BOINC_DIAG_REDIRECTSTDERR           0x00000020L
#define BOINC_DIAG_REDIRECTSTDOUT           0x00000040L
#define BOINC_DIAG_REDIRECTSTDERROVERWRITE  0x00000080L
#define BOINC_DIAG_REDIRECTSTDOUTOVERWRITE  0x00000100L
#define BOINC_DIAG_TRACETOSTDERR            0x00000200L
#define BOINC_DIAG_TRACETOSTDOUT            0x00000400L
```

The flags are as follows. Applications are advised to use at least BOINC_DIAG_DUMPCALLSTACKENABLED, BOINC_DIAG_REDIRECTSTDERR, and BOINC_DIAG_TRACETOSTDERR.

BOINC_DIAG_DUMPCALLSTACKENABLED	If the application crashes, write a symbolic call stack to stderr. If you use this in a Windows app, you must include lib/stackwalker_win.cpp in the compile, and you must include the .pdb (symbol) file in your app version bundle.
BOINC_DIAG_HEAPCHECKENABLED	Check the integrity of the malloc heap every N allocations. (N is line 120 in diagnostics.C; default 1024).
BOINC_DIAG_MEMORYLEAKCHECKENABLED	When process exits, write descriptions of any outstanding memory allocations to stderr.
BOINC_DIAG_ARCHIVESTDERR	Rename stderr.txt to stderr.old on startup.
BOINC_DIAG_ARCHIVESTDOUT	Rename stdout.txt to stdout.old on startup.
BOINC_DIAG_REDIRECTSTDERR	Redirect stderr to stderr.txt.
BOINC_DIAG_REDIRECTSTDOUT	Redirect stdout to stdout.txt.
BOINC_DIAG_REDIRECTSTDERROVERWRITE	Overwrite stderr.txt (default is to append).
BOINC_DIAG_REDIRECTSTDOUTOVERWRITE	Overwrite stdout.txt (default is to append).
BOINC_DIAG_TRACETOSTDERR	Write TRACE macros to stderr (Windows specific).
BOINC_DIAG_TRACETOSTDOUT	Write TRACE macros to stdout (Windows specific).

17. The BOINC graphics API

BOINC applications can optionally provide graphics, which are displayed either in an application window or in a full-screen window (when the BOINC screensaver is selected).

You are encouraged to implement graphics using OpenGL. This makes it easy for your application to show graphics on all platforms.

Integrated graphics

Graphics can either be integrated in your main application or generated by a separate program. The integrated approach is recommended, and we'll describe it first. In this approach, instead of boinc_init(), an application calls

```
#if defined(_WIN32) || defined(__APPLE__)
    retval = boinc_init_graphics(worker);
#else
    retval = boinc_init_graphics_lib(worker, argv[0]);
#endif
```

where `worker()` is the main function of your application. Your application must supply rendering and input-handling functions (see below).

These functions creates a **worker thread** that runs the main application function. The original thread becomes the **graphics thread**, which handles GUI events and does rendering.

On Unix, your graphics code must be put in a separate shared library (.so) file. This is because Unix hosts may not have the needed libraries (OpenGL, GLUT, X11). If an application is linked dynamically to these libraries, it will fail on startup if the libraries are not present. On the other hand, if an application is linked statically to these libraries, graphics will be done very inefficiently on most hosts.

The shared library must have the same name as the executable followed by '.so'. It must be linked with libboinc_graphics_impl.a, with your rendering and input-handling functions, and (dynamically) with glut and opengl. You must bundle the main program and the shared library together as a multi-file application version. Unix/Linux applications that use graphics should compile all files with -D_REENTRANT, since graphics uses multiple threads.

The BOINC example application uses this technique, and shows the Makefile command that are needed to produce the shared library on Unix.

Rendering and input-handling functions

Programs that use integrated graphics must supply the following functions:

```
    void app_graphics_render(int xs, ys, double time_of_day);
```

This will be called periodically in the graphics thread. It should generate the current graphic. `xs` and `ys` are the X and Y sizes of the window, and `time_of_day` is the relative time in seconds. Applications that don't do graphics must also supply a dummy `app_graphics_render()` to link with the API.

```
    void app_graphics_init();
```

This is called in the graphics thread when a window is created. It must make any calls needed to initialize graphics in the window.

```
    void app_graphics_resize(int x, int y);
```

Called when the window size changes.

```
    void app_graphics_reread_prefs();
```

This is called, in the graphics thread, whenever the user's project preferences change. It can call

```
    boinc_parse_init_data_file();
    boinc_get_init_data(APP_INIT_DATA&);
```

to get the new preferences.

The application must supply the following input-handling functions:

```
void boinc_app_mouse_move(
    int x, int y,       // new coords of cursor
    int left,          // whether left mouse button is down
    int middle,
    int right
);

void boinc_app_mouse_button(
    int x, int y,       // coords of cursor
```

```
    int which,          // which button (0/1/2)
    int is_down        // true iff button is now down
);

void boinc_app_key_press(
    int, int            // system-specific key encodings
)

void boinc_app_key_release(
    int, int            // system-specific key encodings
)
```

Limiting frame rate

The following global variables control frame rate:

boinc_max_fps is an upper bound on the number of frames per second (default 30).

boinc_max_gfx_cpu_frac is an upper bound on the fraction of CPU time used for graphics (default 0.5).

Support classes

Several graphics-related classes were developed for SETI@home. They may be of general utility.

REDUCED_ARRAY
: Represents a two-dimensional array of data, which is reduced to a smaller dimension by averaging or taking extrema. Includes member functions for drawing the reduced data as a 3D graph in several ways (lines, rectangles, connected surface).

PROGRESS and PROGRESS_2D
: Represent progress bars, depicted in 3 or 2 dimensions.

RIBBON_GRAPH
: Represents of 3D graph of a function of 1 variable.

MOVING_TEXT_PANEL
: Represents a flanged 3D panel, moving cyclically in 3 dimentions, on which text is displayed.

STARFIELD
: Represents a set of randomly-generated stars that move forwards or backwards in 3 dimensions.

TEXTURE_DESC
: Represents an image (JPEG, Targa, BMP, PNG, or RGB) displayed in 3 dimensions.

The file api/txf_util.C has support functions from drawing nice-looking 3D text.

Static graphics

An application can display a pre-existing image file (JPEG, GIFF, BMP or Targa) as its graphic. This is the simplest approach since you don't need to develop any code. You must include the image file with each workunit. To do this, link the application with api/static_graphics.C (edit this file to use your filename). You can change the image over time, but you must change the (physical, not logical) name of the file each time.

Graphics in a separate program

In this approach, an application bundles a 'main program' and a 'graphics program'. The main program executes the graphics program, and kills it when done. The main and graphics programs typically communicate using shared memory; you can use the functions in boinc/lib/shmem.C for this.

The main program should initialize using

```
int boinc_init_options_graphics(BOINC_OPTIONS&, WORKER_FUNC_PTR worker);
```

The graphics application can be implemented using the BOINC framework, in which case it must initialize with

```
int boinc_init_options_graphics(BOINC_OPTIONS&, NULL);
```

and supply rendering and input-handling functions.

Either the graphics or the main program can handle graphics messages from the core client. It's easiest to have the graphics program handle them; if the main program handles them, it must convey them to the graphics program.

18. Trickle message API

The interface for trickle messages includes both client-side and server-side components.

Client-side API

To send a trickle-up message, call

```
int boinc_send_trickle_up(char* variety, char* text)
```

To receive a trickle-down message, call

```
int boinc_receive_trickle_down(char* buf, int len)
```

This returns true (nonzero) if there was a message.

Server-side API

To handle trickle-up messages, use a 'trickle_handler' daemon. This is a program, based on sched/trickle_handler.C, linked with a function

```
int handle_trickle(MSG_FROM_HOST&);

struct MSG_FROM_HOST {
    int create_time;
    int hostid;
    char variety[256];              // project-defined; what kind of msg
    char xml[MSG_FROM_HOST_BLOB_SIZE];
};
```

This function should return zero if the message was handled successfully; otherwise it will be retried later. The 'hostid' field identifies the host from which the message was sent. The daemon must be passed a '-variety X' command-line argument, telling it what kind of messages to handle. The daemon should be specified in the project configuration file.

To send send trickle-down messages (from a trickle handler daemon or other program) you must insert a record in the 'msg_to_host' table. From C/C++, this is done as follows:

```
DB_MSG_TO_HOST mth;

mth.clear();
mth.create_time = time(0);
mth.hostid = hostid;
sprintf(mth.xml,
    "<trickle_down>\n"
    "   <result_name>%s</result_name>\n"
    "   ...\n"
    "</trickle_down>\n",
    ...
);
retval = mth.insert();
```

To send trickle-down messages, a project must include the line

```
<msg_to_host/>
```

in the configuration (config.xml) file.

19. Intermediate upload

Long-running applications can upload particular output files before the result as a whole is finished. To initiate the upload of an output file, call

```
extern int boinc_upload_file(std::string& name);
```

where 'name' is the logical name of the file. The application cannot modify the file after making this call.

To check on the status of a file being uploaded, call

```
extern int boinc_upload_status(std::string& name);
```

This will return zero if the upload of the file is finished successfully.

20. Building BOINC applications

When building a BOINC application, you typically

- want versions for several platforms (Windows, Mac OS X, Linux);
- want each version to run on a wide range of volunteered computers (e.g. the Linux/x86 version should run on Linux systems of many ages and types).

Achieving these goals can be challenging. However, BOINC provides tools and example files that simplify the task.

Whether you're creating a new application or adapting an existing application to BOINC, we recommend that you start by building the BOINC sample application **upper_case** on all the platforms you want to support. When this is done, you can use the sample application and its associated project files as a basis for your own application.

The first step in building the sample application on a given host is to check out the boinc and boinc_samples modules on that host. Put them in the same parent directory.

Windows

We use Windows XP for builds (other Windows versions may work too).

Microsoft Visual Studio

Go to boinc_samples/win_build. If you're using Visual Studio 2005, open samples.sln. If you're using Visual Studio 2003, open samples_2003.sln. Select 'Build/Build Solution', or hit F7. That's it!

There is a free version, Visual C++ 2005 Express Edition; we think (but haven't verified) that it works with the BOINC project file.

If you use your own project file:

- It must compile BOINC .C files as C++. To do this: in Properties, see C/C++ -> Advanced -> Compile As.
- If your application uses graphics, go to Properties -> Linker -> Input. In Delay Loaded DLLs add

  ```
  GDI32.DLL;OPENGL32.DLL;GLU32.DLL
  ```

  and in Additional Dependencies add

  ```
  delayimp.lib
  ```

- Configure your project to use ANSI strings rather than Unicode.

Cygwin

TODO: links to required software?

TODO: develop a Makefile for building uppercase under Cygwin. It can assume that the libraries in boinc/lib

and boinc/api have already been built. Verify that _autosetup/configure/make work in boinc/.

Information from CERN is here.

Dev-C++

Dev-C++ is an open-source development environment based on the GCC compilers.

TODO: develop a project file for Dev-C++. It should be similar to the Visual Studio project file (i.e. it should include what it needs from boinc/).

Mac OS X

Instructions for building applications on Mac OS X are here.

Linux/x86

If you build an application on a recent Linux distribution, it won't run on older Linux distributions because of library incompatibilities. The only solution we've discovered is to build applications on a host with an old Linux and an old gcc. Setting up such a host, however, is a giant pain.

Our recommended approach is not to build directly on a Linux host, but rather:

- Install Microsoft Virtual PC 2004 on a Windows XP machine (preferably one that's fast and has lots of disk).
- Download the Compatibility virtual machine module. This is a Debian Linux system with the appropriate (old) version of gcc.
- Run this system under Virtual PC.
- Install the boinc and boinc_samples modules in the same parent directory on the virtual host.
- Build the BOINC libraries: go to the boinc/directory and type

  ```
  _autosetup
  ./configure --disable-client --disable_server
  make
  ```

  Do not 'make install'.
- go to the boinc_samples/uppercase directory and type

  ```
  ln -s `g++ -print-file-name=libstdc++.a`
  make
  ```

Other UNIX systems

- Software prerequisites
- configure/build

GLUT notes

Freeglut 2.2, freeglut 2.4 and OpenGL Utility Toolkit (GLUT) libraries are supported.

X11 notes

To get the X11 support, select the relevant options when you're installing Linux, or (Redhat) go to System Settings/Add Software.

21. Example applications

BOINC provides several example applications. See the instructions for building BOINC applications.

The example applications are:

- **upper_case**: a full-featured example BOINC application. The application does things (like checkpointing and graphics) that can be tricky or confusing. You can use it as a template for your own BOINC application; just rip out the computation part (which is trivial) and replace it with your code.

  You can run the application standalone. Create a file 'in' in the directory where you run it; it will convert it to upper case and write it to 'out', and use 20 seconds of CPU time. The graphics show a bouncing 3D ball. If you copy the files boinc/txf/Helvetica.txf and boinc_samples/uppercase/logo.jpg to the directory where it runs, you'll also see an image and some nice-looking text (thanks to Tolu Aina for the latter).

- **wrapper**: used to support legacy applications.
- **worker**: a representative legacy application (i.e. it doesn't use the BOINC API or runtime library). Used for testing wrapper.
- **sleeper**: test application for non-CPU-intensive projects (used for testing the BOINC core client).

22. Application development tips

Cross-platform functions

Most POSIX calls are supported on Unix and Windows. For areas that are different (e.g. scanning directories) BOINC supplies some generic functions with implementations for all platforms. Similar code may be available from other open-source projects.

LIST THEM

Windows-specific issues

- The set of 'standard' DLL differs somewhat among 9X/NT/2000/XP. To avoid crashing because a DLL is missing, call ::LoadLibrary() and then get function pointers.
- Visual Studio: set 'Create/Use Precompiled Header' to 'Automatically Generate' (/YX) in C/C++ Precompiled Header project properties.
- Visual Studio: change 'Compile As' to 'Compile as C++ Code (/TP)' in C/C++ 'Compile As' project properties.

Cross-language issues

The BOINC API is implemented in C++. Information about using it from C and FORTRAN is here.

Compression

If you release new versions frequently, have a large executable, and want to conserve server bandwidth, you may want to compress your executable. The best way to do this is with Ultimate Packer for eXecutables (UPX).

23. Application debugging

Some suggestions for debugging applications:

Standalone mode

When you have built your application and linked it with the BOINC libraries, you can run it in 'standalone

mode' (without a BOINC core client present). To do this, put instances of all input files in the same directory. (with the proper logical, not physical, names). The application should run and produce output files (also with their logical names). You can run the program under a debugger.

When you run an application in standalone mode, the BOINC API will recognize this and take it into account. A couple of things to note:

- If your application does graphics, it will open a graphics window. Closing this window will exit your application.
- boinc_time_to_checkpoint() will always return false, so your application will never checkpoint.

Using the anonymous platform mechanism

Once your application works in standalone mode you'll want to run it from the BOINC core client. This will exercise the various forms of interaction with the core client.

To get this process started, create a test project, add an application version and some work, then run the core client. It will download everything and run your application, which will possibly crash.

At this point you'll want to start experimenting with your application. It would be very tedious to create a new application version for each change. It's far easier to use BOINC's anonymous platform mechanism. To do this:

- Following the directions, create a file 'app_info.xml' in the client's project_* directory, with the appropriate name and version number of your application.
- Each time your build a new version of your application, copy the executable into the project_* directory, making sure it has the appropriate name. Then restart the core client.

On Unix, it's possible to attach a debugger to a running process. Use 'ps' to find the process ID of your application, then something like

```
gdb exec_filename PID
```

to attach a debugger to it.

Getting and deciphering stack traces

Once your application is working on your own computers, you're ready to test it with outside computers (alpha testers initially). It may crash on some computers, e.g. because their software or hardware is different from yours. You'll get some information back in the stderr_txt field of the results. If your application called boinc_init_diagnostics() with the BOINC_DIAG_DUMPCALLSTACKENABLED flag set, and you included symbols, hopefully you'll get symbolic stack traces.

To decipher a Windows stack trace go here.

Otherwise, you should at least get numeric (hex) stack traces. You can decipher these by running a symbolic debugger with an unstripped version and typing in the hex addresses. See http://developer.apple.com/technotes/tn2004/tn2123.html#SECNOSYMBOLS

24. FORTRAN applications

Windows: cygwin

Include the file 'boinc_api_fortran.C' in the api/Makefile.am, but comment out the 'zip' calls, to avoid the linking with 'libboinc_zip.a'

To link it is necessary to include the 'winmm.dll' library (-lwinmm).

Windows: Visual Studio

2004-06-16 note: this page is outdated; will update (functions are now declared `extern"C"` so no C++ mangling is done; there is a boinc_api_fortran.C wrapper) -- quarl@ssl

Note: a working example similar to the following (based on outdated BOINC code) is here; see also its README.

Start by creating a new FORTRAN project. Add all the FORTRAN specific files, then add all the files needed for the BOINC library (e.g. boinc_api.C). Make sure that BOINC and the FORTRAN files are compiled using the same type of standard libraries. i.e. if the BOINC is compiled with the debug multithreaded DLL libraries, make sure the FORTRAN files are compiled with the DLL setting.

For every BOINC function you want to call from fortran you must add an interface and subroutine:

```
INTERFACE
  SUBROUTINE boinc_finish(status)
  END SUBROUTINE boinc_finish
END INTERFACE
```

Remember to declare the type of arguments. INTEGER status

You must then tell the compiler that the function you are interfacing is a C routine. You do this by adding the statement:

```
 !DEC$ ATTRIBUTES C :: boinc_finish
```

Because BOINC is compiled as C++ files the FORTRAN compiler will not be able to find the standard function name in the object file, you therefore have to add an alias for the function giving the real function name:

```
 !DEC$ ATTRIBUTES ALIAS : '?boinc_finish@@YAHH@Z' :: boinc__finish
```

This function name can be found in the object file. Go to your compile directory and run dumpbin.

```
c:\fortranproject\Release>dumpbin /symbols boinc_api.obj
```

this will give you a list of symbols, where you can find the real functionname.

The interface will end up looking like this:

```
INTERFACE
  SUBROUTINE boinc_finish(status)
    !DEC$ ATTRIBUTES C :: boinc_finish
    !DEC$ ATTRIBUTES ALIAS : '?boinc_finish@@YAHH@Z' :: boinc__finish
    INTEGER status
  END SUBROUTINE boinc_finish
END INTERFACE
```

You can now call the BOINC function in FORTRAN.

```
call boinc_finish(0)
```

25. Legacy applications

A **legacy application** is one for which an executable is available, but not the source code. Therefore it cannot use the BOINC API and runtime system. However, such applications can be run using BOINC. Here's an example:

- Compile the program 'worker' from the boinc_samples tree, producing (say) 'worker_5.10_windows_intelx86.exe'. This is the legacy app. If reads from stdin and writes to stdout; it also opens and reads a file 'in', and opens and writes a file 'out'. It takes one command-line argument: the number of CPU seconds to use.
- Compile the program 'wrapper' from the boinc_samples tree, producing (say) 'wrapper_5.10_windows_intelx86.exe'. This program executes your legacy application, and acts as a proxy for it (to report CPU time etc.).
- Create an application named 'worker', and a corresponding directory 'project/apps/worker'. In this

directory, create a directory 'wrapper_5.10_windows_intelx86.exe'. Put the files 'wrapper_5.10_windows_intelx86.exe', and 'worker_5.10_windows_intelx86.exe' there.

- In the same directory, create a file 'job.xml=job_1.12.xml' (1.12 is a version number) containing

```
<job_desc>
    <task>
        <application>worker_5.10_windows_intelx86.exe</application>
        <stdin_filename>stdin</stdin_filename>
        <stdout_filename>stdout</stdout_filename>
        <command_line>10</command_line>
    </task>
</job_desc>
```

This file is read by 'wrapper'; it tells it the name of the legacy program, what files to connect to its stdin/stdout, and a command-line argument.

- Create a workunit template file

```
<file_info>
    <number>0</number>
</file_info>
<file_info>
    <number>1</number>
</file_info>
<workunit>
    <file_ref>
        <file_number>0</file_number>
        <open_name>in</open_name>
        <copy_file/>
    </file_ref>
    <file_ref>
        <file_number>1</file_number>
        <open_name>stdin</open_name>
    </file_ref>
    <rsc_fpops_bound>1000000000000</rsc_fpops_bound>
    <rsc_fpops_est>1000000000000</rsc_fpops_est>
</workunit>
```

and a result template file

```
<file_info>
    <name><OUTFILE_0/></name>
    <generated_locally/>
    <upload_when_present/>
    <max_nbytes>5000000</max_nbytes>
    <url><UPLOAD_URL/></url>
</file_info>
<file_info>
    <name><OUTFILE_1/></name>
    <generated_locally/>
    <upload_when_present/>
    <max_nbytes>5000000</max_nbytes>
    <url><UPLOAD_URL/></url>
</file_info>
<result>
    <file_ref>
        <file_name><OUTFILE_0/></file_name>
        <open_name>out</open_name>
        <copy_file/>
    </file_ref>
    <file_ref>
        <file_name><OUTFILE_1/></file_name>
        <open_name>stdout</open_name>
    </file_ref>
</result>
```

Note that the files opened directly by the legacy program must have the <copy_file> tag.

- Run update_versions to create an app version.
- Run a script like

```
cp download/input `bin/dir_hier_path input`
cp download/input2 `bin/dir_hier_path input2`

bin/create_work -appname worker -wu_name worker_nodelete \
-wu_template templates/worker_wu \
-result_template templates/worker_result \
input input2
```

to generate a workunit.

Notes:

- This requires version 5.5 or higher of the BOINC core client.
- Multiple tasks per job is not implemented yet. Future versions of wrapper may allow you to run multiple applications in sequence (as specified in the job.xml file).
- TODO: provide a way for projects to supply an animated GIFF which is shown (with user/team credit text) as screensaver graphics.

To understand how all this works: at the beginning of execution, the file layout is:

Project directory	slot directory
job_1.12.xml	in (copy of project/input)
input	job.xml (link to project/job_1.12.xml)
input2	stdin (link to project/input2)
worker_5.10_windows_intelx86.exe	stdout (link to project/worker_nodelete_0)
wrapper_5.10_windows_intelx86.exe	worker_5.10_windows_intelx86.exe (link to project/worker_5.10_windows_intelx86.exe)
	wrapper_5.10_windows_intelx86.exe (link to project/wrapper_5.10_windows_intelx86.exe)

The wrapper program executes the worker, connecting its stdin to project/input2 and its stdout to project/worker_nodelete_0. The worker program opens 'in' for reading and 'out' for writing.

When the worker program finishes, the wrapper sees this and exits. Then the the BOINC core client copies slot/out to project/worker_nodelete_1.

26. Compound applications

Compound applications

A **compound application** consists of a **main program** and one or more **worker programs**. The main program executes the worker programs in sequence, and maintains a 'main state file' that records which worker programs have completed. The main program assigns to each worker program a subrange of the overall 'fraction done' range of 0..1. For example, if there are two worker programs with equal runtime, the first would have range 0 to 0.5 and the second would have range 0.5 to 1.

The BOINC API provides a number of functions, and in developing a compound application you must decide whether these functions are to be performed by the main or worker program. The functions are represented by flags in the BOINC_OPTIONS structure. Each flag must be set in either the main or worker programs, but not both.

```
struct BOINC_OPTIONS {
    int main_program;
    int check_heartbeat;
    int handle_trickle_ups;
    int handle_trickle_downs;
    int handle_process_control;
    int handle send_status_msgs;
    int direct_process_action;
};

int boinc_init_options(BOINC_OPTIONS*);
```

main_program	Set this in the main program.
check_heartbeat	If set, the program monitors 'heartbeat' messages from the core client. If the heartbeat stops, the result depends on the direct_process_action flag (see below).
handle_trickle_ups	If set, the program can send trickle-up messages.
handle_trickle_downs	If set, the program can receive trickle-down messages.

handle_process_control	If set, the program will handle 'suspend', 'resume', and 'quit' messages from the core client. The action depends on the direct_process_action flag.
send_status_msgs	If set, the program will report its CPU time and fraction done to the core client. Set in worker programs.
direct_process_action	If set, the program will respond to quit messages and heartbeat failures by exiting, and will respond to suspend and resume messages by suspending and resuming. Otherwise, these events will result in changes to the BOINC_STATUS structure, which can be polled using boinc_get_status().
all_threads_cpu_time	If set, the CPU of all threads (not just the worker thread) will be counted. For apps that do computation in more than one thread.

Typical main program logic is:

```
BOINC_OPTIONS options;

options.main_program = true;
...
boinc_init_options(&options)
read main state file
for each remaining worker program:
    APP_INIT_DATA aid;
    boinc_get_init_data(aid);
    aid.fraction_done_start = x
    aid.fraction_done_end = y
    boinc_write_init_data_file(aid)
    run the app
    wait for the app to finish
        poll
    write main state file
    if last app:
        break
    boinc_parse_init_data_file()    // reads CPU time from app_init.xml file
boinc_finish()
```

where x and y are the appropriate fraction done range limits.

Typical worker program logic is:

```
BOINC_OPTIONS options;

options.main_program = false;
options.send_status_msgs = true;
...
boinc_init_options(&options);
...
do work, calling boinc_fraction_done() with values from 0 to 1,
and boinc_time_to_checkpoint(), occasionally
...
boinc_finish();        // this writes final CPU time to app_init.xml file
```

If the graphics is handled in a program that runs concurrently with the worker programs, it must also call `boinc_init_options()`, typically with all options false, then `boinc_init_graphics()`, and eventually `boinc_finish()`.

If the main program is responsible for reporting application status to the core client, it should periodically call

```
    boinc_report_app_status(
        double cpu_time,                // CPU time since start of WU
        double checkpoint_cpu_time,     // CPU time at last checkpoint
        double fraction_done
    );
```

27. Setting up a BOINC server

You can potentially use any Unix system as a BOINC server. Our recommendations are:

- **Hardware**: use a host with good CPU capacity (dual Xeon or Opteron), at least 2 GB of RAM, and at least 40 GB of free disk space. Do whatever you can to make it highly reliable (UPS power supply, RAID disk configuration, hot-swappable spares, tempurature-controlled machine room, etc.). If you anticipate a high-traffic project, use a machine whose RAM capacity is 8 GB or more, and that uses 64-bit processors.

  If your server capacity is exceeded, you can increase capacity using multiple server hosts. However, we recommend that you not do this initially. In most cases one host is enough.

- **Software**: use a recent Linux release.

Groups and permissions

We recommend that you create a separate user and group for the BOINC server, and add the web-server user to this group. Details are here.

Installing BOINC software

- Download and install whatever software prerequisites are needed on your system.
- Download the BOINC software.
- Configure and build the BOINC software.

Operating system configuration

Some parts of the BOINC server (the feeder and scheduling server) use shared memory. Hosts where these run must have shared memory enabled, with a maximum segment size of at least 32 MB. How to do this depends on the operating system; some information is here.

MySQL notes

- After installing and running the server, grant permissions for your own account and for the account under which Apache runs ('nobody' in the following; may be different on your machine). All mysql accounts should be password protected including root.

```
mysql -u root
grant all on *.* to yourname@localhost identified by 'password';
grant all on *.* to yourname identified by 'password';
grant all on *.* to nobody@localhost identified by 'password';
grant all on *.* to nobody identified by 'password';
```

- Set your PATH variable to include MySQL programs (typically /usr/local/mysql and /usr/local/mysql/bin).
- You'll need to back up your database. Generally this requires stopping the project, making a copy or snapshot, and restarting. An example is here.
- BOINC gets MySQL compiler and linker flags from a program called `mysql_config` which comes with your MySQL distribution. This sometimes references libraries that are not part of your base system installation, such as `-lnsl` or `-lnss_files`. You may need to install additional packages (often you can use something called 'mysql-dev' or 'mysql-devel') or fiddle with Makefiles.
- MySQL can be the bottleneck in a BOINC server. To optimize its performance, read about configuring MySQL for BOINC.
- Notes on running MySQL on a cluster.

MySQLclient notes

- Configure mysql with the --enable-thread-safe-client switch.
- Set your LD_LIBRARY_PATH to refer to the correct library.

Apache notes

In httpd.conf, set the default MIME type as follows (otherwise you'll get file upload signature verification errors):

```
DefaultType application/octet-stream
```

PHP notes

- Make sure 'magic quotes' are enabled (this is the default). The file /etc/php.ini should contain

  ```
  magic_quotes_gpc = On
  ```

- By default, BOINC uses PHP's `mail` function to send email to participants. This uses sendmail. If this doesn't work, you can use PHPMailer instead, which is a very flexible mail-sending mechanism. To do this:
  - Download PHPMailer and put it under PROJECT/html/inc/phpmailer.
  - Set the following variables in your PROJECT/html/project/project.inc file (substitute your own values):

    ```
    = true;
    = "xxx.xxx.xxx";
    = "smtp";
    ```

28. What is a project?

A BOINC project consists of the following components:

- A database
- A directory structure
- A configuration file, which specifies options, daemons, and periodic tasks.

Multiple BOINC projects can exist on the same host. This can be handy for creating separate projects for testing and debugging.

The easiest way to create a project is with the make_project script, which creates skeletal versions of the above components.

A project must be **stopped** when maintenance activities (e.g. changing the configuration file or database) are being performed. This is done using project control scripts.

The master URL

Each project is publicly identified by a **master URL**. The **master page** at this URL has two functions.

- It is the home page of the project; when viewed in a browser it describes the project and contains links for registering and for downloading the core client.
- It contains XML elements of the form

  ```
  <scheduler>http://host.domain.edu/cgi/scheduler</scheduler>
  <scheduler>http://host2.domain.edu/cgi/scheduler</scheduler>
  ```

  that give the URLs of the project's scheduling servers. These tags can be embedded within HTML comments. The BOINC core client reads and parses the master page to locate scheduling servers. If at any point it is unable to connect to any scheduling server for a project, it rereads the master page. This mechanism lets a project move or add scheduling servers.

make_project creates a master page in project/html/user/index.php. This file includes the file 'schedulers.txt', which contains the list of <scheduler>elements.

29. The BOINC database

BOINC stores information in a MySQL database. The main tables are:

platform	Compilation targets of the core client and/or applications.
app	Applications. The core client is treated as an application; its name is 'core_client'.
app_version	Versions of applications. Each record includes a URL for downloading the executable, and the MD5 checksum of the executable.
user	Describes users, including their email address, name, web password, and authenticator.
host	Describes hosts.
workunit	Describes workunits. The input file descriptions are stored in an XML document in a blob field. Includes counts of the number of results linked to this workunit, and the numbers that have been sent, that have succeeded, and that have failed.
result	Describes results. Includes a 'state' (whether the result has been dispatched). Stores a number of items relevant only after the result has been returned: CPU time, exit status, and validation status.

The database is created by the make_project script. Normally you don't have to directly examine or manipulate the database. If you need to, you can use the MySQL command-line interpreter or BOINC's administrative web interface.

30. Server directory structure

The directory structure for a typical BOINC project looks like:

```
PROJECT/
    apps/
    bin/
    cgi-bin/
    log_HOSTNAME/
    pid_HOSTNAME/
    download/
    html/
        inc/
        ops/
        project/
        stats/
        user/
        user_profile/
    keys/
    upload/
```

where PROJECT is the name of the project and HOSTNAME is the server host. Each project directory contains:

- apps: application and core client executables
- bin: server daemons and programs.
- cgi-bin: CGI programs
- log_HOSTNAME: log output
- pid_HOSTNAME: lock files, pid files
- download: storage for data server downloads.
- html: PHP files for public and private web interfaces
- keys: encryption keys
- upload: storage for data server uploads.

The upload and download directories may contain large numbers (millions) of files. For efficiency they are normally organized as a hierarchy of subdirectories.

31. The project configuration file

A project's configuration is described by a file named **config.xml** in the project's directory. This file is

created, with default values, by the make_project script. However, you will need to change or add some items during the life of your project.

The format of config.xml file is:

```
<boinc>
  <config>
    [ configuration options ]
  </config>
  <daemons>
    [ list of daemons ]
  </daemons>
  <tasks>
    [ list of periodic tasks ]
  </tasks>

</boinc>
```

Details on:

- Configuration options
- Daemons
- Periodic tasks

32. Project options

The following elements in the <config> section of your config.xml file control various aspects of your project.

Hosts, directories, and URLs

(These are created by make_project; normally you don't need to change them.)

```
<master_url>            URL                     </master_url>
<long_name>             name                    </long_name>
<host>                  project.hostname.ip   </host>
<db_name>               databasename          </db_name>
<db_host>               database.host.ip      </db_host>
<db_user>               database_user_name    </db_user>
<db_passwd>             database_password     </db_passwd>
<shmem_key>             shared_memory_key     </shmem_key>
<download_url>          http://A/URL          </download_url>
<download_dir>          /path/to/directory    </download_dir>
<download_dir_alt>      /path/to/directory    </download_dir_alt>
<uldl_dir_fanout>       N                     </uldl_dir_fanout>
<upload_url>            http://A/URL          </upload_url>
<upload_dir>            /path/to/directory    </upload_dir>
<cgi_url>               http://A/URL          </cgi_url>
<!-- optional; defaults as indicated: -->
<project_dir>  ../      </project_dir>  <!-- relative to location of 'start' -->
    <bin_dir>      bin      </bin_dir>      <!-- relative to project_dir -->
    <cgi_bin_dir>  cgi-bin  </cgi_dir>      <!-- relative to project_dir -->
[ <sched_lockfile_dir> path </sched_lockfile_dir> ]
```

host	name of project's main host, as given by Python's socket.hostname(). Daemons and tasks run on this host by default.
db_name	Database name
db_host	Database host machine
db_user	Database user name
db_passwd	Database password
shmem_key	ID of scheduler shared memory. Must be unique on host.
download_url	URL of data server for download
download_dir	absolute path of download directory

download_dir_alt	absolute path of old download directory (see Hierarchical upload/download directories)
upload_url	URL of file upload handler
uldl_dir_fanout	fan-out factor of upload and download directories (see Hierarchical upload/download directories)
upload_dir	absolute path of upload directory
cgi_url	URL of scheduling server
sched_lockfile_dir	Enables scheduler locking (recommended) and specifies directory where scheduler lockfiles are stored. Must be writable to the Apache user.

Web site features

```
[ <profile_screening/> ]
[ <show_results/> ]
```

profile_screening	If present, don't show profile pictures until they've been screened and approved by project admins.
show_results	Enable web site features that show results (per user, host, etc.)

Miscellaneous

```
[ <disable_account_creation/> ]
[ <min_passwd_length> N </min_passwd_length> ]
```

disable_account_creation	If present, disallow account creation
min_passwd_length	Minimum length of user passwords. Default is 6.

Client control

```
[ <verify_files_on_app_start/> ]
[ <symstore>URL</symstore> ]
[ <min_core_client_version_announced> N </min_core_client_version_announced> ]
[ <min_core_client_upgrade_deadline> N </min_core_client_upgrade_deadline> ]
[ <non_cpu_intensive> 0|1 </non_cpu_intensive> ]
```

verify_files_on_app_start	Before starting or restarting an app, check contents of input files and app version files by either MD5 or digital signature check. Detects user tampering with file (but doesn't really increase security, since user could also change MD5s or signatures in client state file).
symstore	URL of your project's symbol store, used for debugging Windows applications.
min_core_client_version_announced	Announce a new version of the BOINC core client, which in the future will be the minimum required version. In conjunction with the next tag, you can warn users with version below this to upgrade by a specified deadline. Example value: 419.
min_core_client_upgrade_deadline	Use in conjunction with the previous tag. The value given here is the Unix epoch returned by time(2) until which hosts can update their core client. After this time, they may be shut out of the project. Before this time, they will receive messages warning them to upgrade.

non_cpu_intensive

If this flag is present, the project will be treated specially by the client:

- The client will download one result at a time.
- This result will be executed whenever computation is enabled (bypassing the normal scheduling mechanism).

This is intended for applications that use little CPU time, e.g. that do network or host measurements.

Server logging

```
[ <sched_debug_level> N </sched_debug_level> ]
[ <fuh_debug_level> N </fuh_debug_level> ]
```

sched_debug_level Verbosity level for scheduler log output. 1=minimal, 2=normal (default), 3=verbose.

fuh_debug_level Verbosity level for file upload handler log output. 1=minimal, 2=normal (default), 3=verbose.

Credit

(See also the command-line options of the validator).

```
[ <fp_benchmark_weight> X </fp_benchmark_weight> ]
```

fp_benchmark_weight The weighting given to the Whetstone benchmark in the calculation of claimed credit. Must be in [0 .. 1]. Projects whose applications are floating-point intensive should use 1; pure integer applications, 0. Choosing an appropriate value will reduce the disparity in claimed credit between hosts. The script html/ops/credit_study.php, run against the database of a running project, will suggest what value to use.

Scheduling options and parameters

```
[ <one_result_per_user_per_wu/> ]
[ <max_wus_to_send> N </max_wus_to_send> ]
[ <min_sendwork_interval> N </min_sendwork_interval> ]
[ <daily_result_quota> N </daily_result_quota> ]
[ <ignore_delay_bound/> ]
[ <dont_generate_upload_certificates/> ]
[ <locality_scheduling/> ]
[ <locality_scheduling_wait_period> N </locality_scheduling_wait_period> ]
[ <min_core_client_version> N </min_core_client_version ]
[ <choose_download_url_by_timezone> 0|1 </choose_download_url_by_timezone> ]
[ <cache_md5_info> 0|1 </cache_md5_info> ]
[ <nowork_skip> 0|1 </nowork_skip> ]
[ <resend_lost_results> 0|1 </resend_lost_results> ]
[ <default_disk_max_used_gb> X </default_disk_max_used_gb> ]
[ <default_disk_max_used_pct> X </default_disk_max_used_pct> ]
[ <default_disk_min_free_gb> X </default_disk_min_used_pct> ]
```

one_result_per_user_per_wu If present, send at most one result of a given workunit to a given user. This is useful for checking accuracy/validity of results. It ensures that the results for a given workunit are generated by **different** users. If you have a validator that compares different results for a given workunits to ensure that they are equivalent, you should probably enable this. Otherwise you may end up validating results from a given user with results from the **same** user.

max_wus_to_send Maximum results sent per scheduler RPC. Helps prevent hosts with trouble from getting too many results and trashing them. But you should set this large enough so that a host which is only connected

	to the net at intervals has enough work to keep it occupied in between connections.
min_sendwork_interval	Minimum number of seconds to wait after sending results to a given host, before new results are sent to the same host. Helps prevent hosts with download or application problems from trashing lots of results by returning lots of error results. But don't set it to be so long that a host goes idle after completing its work, before getting new work.
daily_result_quota	Maximum number of results (per CPU) sent to a given host in a 24-hour period. Helps prevent hosts with download or application problems from returning lots of error results. Be sure to set it large enough that a host does not go idle in a 24-hour period, and can download enough work to keep it busy if disconnected from the net for a few days. The maximum number of CPUS is bounded at four.
ignore_delay_bound	By default, results are not sent to hosts too slow to complete them within delay bound. If this flag is set, this rule is not enforced.
dont_generate_upload_certificates	Don't put upload certificates in results. This makes result generation a lot faster, since no encryption is done, but you lose protection against DoS attacks on your upload servers.
locality_scheduling	When possible, send work that uses the same files that the host already has. This is intended for projects which have large data files, where many different workunits use the same data file. In this case, to reduce download demands on the server, it may be advantageous to retain the data files on the hosts, and send them work for the files that they already have. See Locality Scheduling.
locality_scheduling_wait_period	This element only has an effect when used in conjunction with the previous locality scheduling element. It tells the scheduler to use 'trigger files' to inform the project that more work is needed for specific files. The period is the number of seconds which the scheduler will wait to see if the project can create additional work. Together with project-specific daemons or scripts this can be used for 'just-in-time' workunit creation. See Locality Scheduling.
min_core_client_version	If the scheduler gets a request from a client with a version number less than this, it returns an error message and doesn't do any other processing.
choose_download_url_by_timezone	When the scheduler sends work to hosts, it replaces the download URL appearing in the data and executable file descriptions with the download URL closest to the host's timezone. The project must provide a two-column file called 'download_servers' in the project root directory. This is a list of all download servers that will be inserted when work is sent to hosts. The first column is an integer listing the server's offset in seconds from UTC. The second column is the server URL in the format such as http://einstein.phys.uwm.edu. The download servers must have identical file hierarchies and contents, and the path to file and executables must start with '/download/...' as in 'http://X/download/123/some_file_name'.
cache_md5_info	When creating work, keep a record (in files called foo.md5) of the file length and md5 sum of data files and executables. This can greatly reduce the time needed to create work, if (1) these files are re-used, and (2) there are many of these files, and (3) reading the files from disk is time-consuming.

nowork_skip	If the scheduling server has no work, it replies to RPCs without doing any database access (e.g., without looking up the user or host record). This reduces DB load, but it fails to update preferences when users click on Update. Use it if your server DB is overloaded.
resend_lost_results	If set, and a <other_results> list is present in scheduler request, resend any in-progress results not in the list. This is recommended; it should increase the efficiency of your project. Note that in an ideal world, it would never be necessary to resend results. The first time that they were sent from the server, the client would receive them and no resending would be needed. However it is the experience of several projects that, for reasons that are not well understood, a BOINC client sometimes fails to receive the scheduler reply. This flag addresses that issue: it causes the SAME results to be RESENT by the scheduler, if the client has failed to receive them. It works as follows. In its scheduler request, the BOINC client includes a list of results that it has already received. The scheduler checks these against the database to be sure that the client has received ALL results which should be present. If there are missing results on client, and this flag is set, then those results are RESENT by the scheduler before any new work is sent.
default_disk_max_used_gb	Sets the default value for the disk_max_used_gb preference so its consistent between the scheduler and web pages. The scheduler uses it when a request for work doesn't include preferences, or the preference is set to zero. The web page scripts use it to set the initial value when displaying or editing preferences the first time, or when the user never saved them. Default is 100.
default_disk_max_used_pct	Sets the default value for the disk_max_used_pct preference so its consistent between the scheduler and web pages. The scheduler uses it when a request for work doesn't include preferences, or the preference is set to zero. The web page scripts use it to set the initial value when displaying or editing preferences the first time, or when the user never saved them. Default is 50.
default_disk_min_free_gb	Sets the default value for the disk_min_free_gb preference so its consistent between the scheduler and web pages. The scheduler uses it when a request for work doesn't include preferences. The web page scripts use it to set the initial value when displaying or editing preferences the first time, or when the user never saved them. Also, the scheduler uses this setting to override any smaller preference from the host, it enforces a 'minimum free disk space' to keep from filling up the drive. Recommend setting this no smaller than .001 (1MB or 1,000,000 bytes). Default is .001.
reliable_time **reliable_min_avg_credit** **reliable_min_avg_turnaround** **reliable_reduced_delay_bound**	These parameters control a mechanism that attempts to send old results (e.g. those whose siblings have timed out or failed) to fast, reliable hosts. This mechanism is used when the age of a workunit exceeds **reliable_time** (typically 2-3X the delay bound). The results are sent to hosts for which expavg_credit/ncpus is at least **reliable_min_avg_credit** and whose average turnaround is at most **reliable_max_avg_turnaround**. The delay bound is multiplied by **reliable_reduced_delay_bound** (typically 0.5 or so).

File deleter options

```
[ <dont_delete_batches/> ]
```

dont_delete_batches If this boolean is set, the file deleter won't delete any files for which the corresponding workunit or result record has a positive value of the the 'batch' field. This lets you keep files on disk until you're done with them. Create workunits with a positive batch number, and zero out (or negate) the batch number when you're done looking at the files (you can do this with a SQL query). If you use this option, replace the indices on file_delete_state with indices on (file_delete_state, batch).

Server status page options

```
[ <www_host>hostname</www_host> ]
[ <sched_host>hostname</sched_host> ]
[ <sched_pid>path</sched_pid> ]
[ <uldl_host>hostname</uldl_host> ]
[ <uldl_pid>path</uldl_pid> ]
[ <ssh_exe>path</ssh_exe> ]
[ <ps_exe>path</ps_exe> ]
```

www_host Host name of web server

sched_host Host name of scheduling server

sched_pid pid file of scheduling server (default: /etc/httpd/run/httpd.pid)

uldl_host Host name of upload/download server

sched_pid pid file of upload/download server (default: /etc/httpd/run/httpd.pid)

ssh_exe path to ssh (default: /usr/bin/ssh)

ps_exe path to ps (which supports "w" flag) (default: /bin/ps)

33. Daemons

Daemons are server programs that normally run continuously. Your project's daemons are described in its config.xml file, with elements of the form:

```
<daemon>
  <cmd> feeder -d 3 </cmd>
  [ <host>host.domain.name</host> ]
  [ <disabled> 0|1 </disabled> ]
  [ <output>filename</output> ]
  [ <pid_file>filename</pid_file> ]
</daemon>
<daemon>
...
</daemon>
```

cmd The command used to start the daemon. Must be a program in the project's bin/ directory.

host Specifies the host on which the daemon should run. The default is the project's main host, as specified in config.xml.

disabled If nonzero, ignore this entry

output Name of output file (in the log_HOSTNAME directory). Defaults to the program name followed by '.log'. If you're running multiple instances of a daemon on one host, you must specify this.

pid_file Name of file used to store the process ID (in the pid_HOSTNAME directory). Defaults to the program name followed by '.pid'. If you're running multiple instances of a daemon on one host, you must specify this.

Daemons are started when you run the bin/start script, and killed (by a SIGHUP signal) when you run

bin/stop.

Typically, this mechanism is used to run work handling daemons. Projects that use trickle-up messages will also need to have a trickle-up handler.

34. Periodic tasks

Periodic tasks are programs that are run periodically. They are executed by the bin/start --cron program, which you should run from cron. (To do this, run crontab and add a line of the form

```
0,5,10,15,20,25,30,35,40,45,50,55 * * * * HOME/projects/PROJECT/bin/start --cron
```

Periodic tasks are short-running, but in case they aren't, the 'start' script detect when an instance is still running and won't start another instance.

Your project's periodic tasks are described in its config.xml file, with elements of the following form:

```
<task>
  <cmd>          get_load        </cmd>
  <output>       get_load.out    </output>
  <period>       5 min           </period>
  [ <host>       host.ip         </host>       ]
  [ <disabled>   1               </disabled>   ]
  [ <always_run> 1               </always_run> ]
</task>
<task>
  <cmd>      echo "HI" | mail root@example.com     </cmd>
  <output>   /dev/null                             </output>
  <period>   1 day                                 </period>
</task>
<task>
...
</task>
```

cmd The command used to perform the task. Must be a program in the project's /bin directory.

You can run PHP scripts as periodic tasks. These scripts must be in the html/ops/ directory, and can be run with a command of the form

```
run_in_ops scriptname
```

The script should be executable, and should have the form

```
#! /usr/bin/env php
<?php
...
?>
    You must specify the output file for such tasks
    (otherwise it will go to run_in_ops.out).
```

host Specifies where the daemon shoulr run. The default is the project's main host, as specified in config.xml.

period The interval between executions, expressed as a single number (in units of minutes) or a number followed by 'seconds', 'minutes', 'hours', or 'days' (may be abbreviated to unique initial string).

output Specifies the output file to output; and by default it is COMMAND_BASE_NAME.out. Output files are in log_X/, where X is the host.

disabled Ignore this entry

always_run Run this task regardless of whether or not the project is enabled (for example, a script that logs the current CPU load of the host machine).

A project newly created by make_project has no periodic tasks. Here are some programs you might want to run as periodic tasks:

db_dump	Write statistics data to XML files for export. Details are here. Recommended period: 1 day.
update_profile_pages.php	Generate HTML files with lists of links to user profiles, organized alphabetically and by country. Recommended period: a few days.
update_stats -update_teams -update_users -update_hosts	Update the recent average credit fields of users, teams, and hosts. This is important if you use send personalized mass emails or reminder emails, or use recent credit to enable message-board posting. Recommended period: every few days.
update_uotd.php	Select a new User of the Day. Period: 1 day.
update_forum_activities.php	Recompute 'popularity' estimages for threads and posts in the Questions and Answers message boards. Recommended period: 1 hour.

35. The make_project script

The `make_project` script creates the server components of a BOINC project. To use it, set up a BOINC server. Then, for example, type:

```
cd tools/
./make_project cplan
```

creates a project with master URL http://<hostname>/cplan/ whose directory structure is rooted at $HOME/projects/cplan.

More specifically, `make_project` does the following:

- Create the project directory and its subdirectories.
- Create the project's encryption keys if necessary. NOTE: before making the project visible to the public, you must move the code-signing private key to a highly secure (preferably non-networked) host, and delete it from the server host.
- Create and initialize the MySQL database
- Copy source and executable files
- Generate the project's configuration file.

The script gives further instructions, namely

- It generates a template Apache config file that you can insert into /etc/apache/httpd.conf (path varies), or Include directly.
- It generates a crontab line to paste.

The command-line syntax is as follows:

```
make_project [options] project_name [ 'Project Long Name ' ]
```

Options are as follows (normally you don't need to include any of them):

directory options	
--project_root	Project root directory path. Default: $HOME/projects/PROJECT_NAME
--key_dir	Where keys are stored. Default: PROJECT_ROOT/keys
--url_base	Determines master URL Default: http://$NODENAME/
--no_query	Accept all directories without yes/no query

--delete_prev_inst	Delete project-root first (from prev installation)
	URL options
--html_user_url	User URL. Default: URL_BASE/PROJECT/
--html_ops_url	Admin URL. Default: URL_BASE/PROJECT_ops/
--cgi_url	CGI URL. Default: URL_BASE/PROJECT_cgi/
	database options
--db_host	Database host. Default: none (this host)
--db_name	Database name. Default: PROJECT
--db_user	Database user. Default: current user
--db_passwd	Database password. Default: None
--drop_db_first	Drop database first (from prev installation)
	debugging options
--verbose={0,1,2}	default: 1
-v	alias for --verbose=2
-h or --help	Show options

36. xadd - tool for adding database items

xadd adds platform and application records to the BOINC database. Information is read from an XML file **project.xml** in the project's root directory. Run `xadd` from the project root directory, i.e.:

```
cd ~/projects/project_name
bin/xadd
```

The contents of **project.xml** should look like this:

```
  <boinc>
    <app>
        <name>setiathome</name>
        <user_friendly_name>SETI@home</user_friendly_name>
        [ <beta>1</beta> ]
    </app>
    ...
    <platform>
        <name>anonymous</name>
        <user_friendly_name>anonymous</user_friendly_name>
    </platform>
    <platform>
        <name>i686-pc-linux-gnu</name>
        <user_friendly_name>Linux/x86</user_friendly_name>
    </platform>
    <platform>
        <name>windows_intelx86</name>
        <user_friendly_name>Windows/x86</user_friendly_name>
    </platform>
    <platform>
        <name>powerpc-apple-darwin</name>
        <user_friendly_name>Mac OS X</user_friendly_name>
    </platform>
    ...
  </boinc>
```

The 'name' of each item should be short and without spaces or other special characters. The 'user friendly name' (shown to end users) has no restrictions. An example project.xml file is in source/tools/.

`xadd` only adds new items; entries that conflict with existing database entries are ignored.

37. Releasing application versions

The `update_versions` script releases new application versions. It creates database entries and copies files to the download directory.

To use:

- If it doesn't already exit, create an directory 'apps' under the project directory, and add an <app_dir> element to config.xml giving the path of the apps directory.
- Create a subdirectory for each application, with the short name of the application. Put new application files here (see below). `update_versions` scans these directories for new application versions.
- From the project's root directory, run bin/update_versions

Single-file application versions

File names must be of the form `NAME_VERSION_PLATFORM[.ext]`, e.g.:

```
boinc_3.17_i686-pc-linux-gnu
astropulse_7.17_windows_intelx86.exe
```

Platform strings must match the names of platforms in the database. If needed, add the platform to the DB.

Multiple-file application versions

Application versions can consist of multiple files, one of which is the main program. To create a multiple-file application version, create a directory with the same name as the main program (of the form NAME_VERSION_PLATFORM[.ext]). and put the files in that directory.

If your application includes executable files other than the main file, make sure that their protection flags include the user execute (u+x) bit.

Passing extra information about files

If a file of the form

```
FILENAME.sig
```

is found, its contents will be used as a digital signature for the corresponding file. Recommended code-signing practices are described here.

If a file of the form

```
FILENAME.file_ref_info
```

is found, its contents will be added to the <file_ref> element describing the file (you can use this for attributes like <copy_file>).

If a file of the form

```
LOGICAL_NAME=PHYSICAL_NAME
```

is found, the given logical and physical names will be used (i.e., the application will be able to access the file by passing the logical name to boinc_resolve_filename()).

38. Project creation cookbook

Make skeletal project

- Install and configure all prerequisite software (follow the directions carefully). Make sure MySQL is configured and running.
- Get the BOINC
- Compile the BOINC software, say into HOME/boinc.
- Run HOME/boinc/tools/make_project
- Append the contents of projects/PROJECT/PROJECT.httpd.conf to httpd.conf and restart Apache.
- Use 'crontab' to insert a cron job to run the project's periodic tasks, e.g.
  0,5,10,15,20,25,30,35,40,45,50,55 * * * * HOME/projects/PROJECT/bin/start --cron
  (if cron cannot run 'start', try using a helper script to set PATH and PYTHONPATH)
- Copy project.xml from HOME/boinc/tools to HOME/projects/PROJECT, edit it to reflect your applications and platforms, and run bin/xadd.
- Edit html/project/project.inc, changing the master URL and copyright holder.
- Protect the html/ops directory (e.g. by putting .htaccess and .htpasswd files there).

Visible result: the project web site is up. The database 'platforms' table has several rows.

Troubleshooting: check the Apache access and error logs.

Create an application version

- Create a BOINC application executable (if you're in a hurry, use the test application).
- Copy the executable to HOME/projects/PROJECTNAME/apps/APPNAME
- cd to HOME/projects/PROJECTNAME
- run bin//update_versions, type y or return.
- run ./stop && ./start

Visible result: the web site's Applications page has an entry.

Create a work unit

- Using a text editor, create a work unit template file and a result template file.
- Run create_work
- Edit config.xml to add <daemon> records for make_work, feeder, transitioner, file_deleter, the trivial validator, and the trivial assimilator. For example

```
<daemon>
    <cmd>validate_test -app appname</cmd>
    <output>validate_test.log</output>
    <pid_file>validate_test.pid</pid_file>
</daemon>
```

Visible result: after a project restart, 'status' shows the above daemon processes running.

Troubleshooting: check the log files of all daemon processes.

Test the system

- Create a client directory (on the same computer or different computer), say HOME/boinc_client. Copy the core client there.
- Using the web interface, create an account on the project.
- Run the core client; enter the project URL and the account key.

Visible result: the client does a stream of work; the web site shows credit accumulating.

Troubleshooting: check the log files of all daemon processes.

Develop back end components

- Write a work generator.
- Write a validator.
- Write an assimilator.
- Edit the configuration file to use these programs instead of the place-holder programs.
- Make sure everything works correctly.

Extras

- Add message board categories: see html/ops/create_forums.php

39. Project control

Project control scripts

The following Python scripts control a project:

bin/start	Start the project: start all daemons, and remove the stop_sched and stop_daemon files (see below).
bin/stop	Stop the project (create the stop_sched and stop_daemon files)
bin/start --cron	If the project is started, perform all periodic tasks that are past due, and start any daemons that aren't running. Otherwise do nothing.
bin/status	Show whether the project is stopped. Show the status of all daemons. Show the status of all periodic tasks (e.g., when they were last executed).

Trigger files

The following files (in the project root directory) can be used to turn off various parts of a project.

stop_sched	Have the scheduler return 'project down' messages.
stop_daemons	Tell all daemon process to exit.
stop_web	Have the web site return 'project down' messages for all functions that require database access.
stop_upload	Have the file upload handler return transient error messages to clients (they'll back off and retry later).

The presence of a file triggers the function. For example, to turn off data-driven web pages, type

```
touch stop_web
```

and to turn them back on, type

```
rm stop_web
```

If the first three files are all present, no database access will occur. You should do this during most maintenance functions (such as upgrading software).

40. Project security

Before creating a BOINC project, read about security issues in volunteer computing. BOINC provides

mechanisms that address the major issues, making volunteer computing safe both for you and for participants.

If you don't use these mechanisms correctly, your project will be vulnerable to a variety of attacks. In the worst case, your project could be used as a vector to distribute malicious software to large numbers of computers. This would be fatal to your project, and would cause serious damage to volunteer computing in general.

We recommend that you do the following:

- Secure each of your server computers as much as possible. Read and implement the UNIX Security Checklist 2.0 from AusCERT and CERT/CC.
- Put all server computers behind a firewall that lets through minimal traffic (e.g., HTTP and SSH where needed).
- Read about MySQL general security guidelines, and make your MySQL server as secure as possible.
- Make sure your application doesn't become infected. Secure your source-code repository, and examine all checkins. If your application uses third-party libraries, make sure they're safe. Read about Secure Programming for Linux and Unix, especially if your application does network communication.
- Use BOINC's code-signing mechanism, and use a disconnected and physically secure code-signing computer.

41. Code signing

BOINC uses digital signatures to allow the core client to authenticate executable files.

It is important that you use a proper code-signing procedure for publicly-accessible projects. If you don't, and your server is broken into, hackers will be able to use your BOINC project to distribute whatever malicious code they want. This could result in the end of your project, and possibly the end of all BOINC projects.

- Choose a computer (an old, slow one is fine) to act as your 'code signing machine'. After being set up, this computer **must remain physically secure and disconnected from the network** (i.e. keep it in a locked room and put duct tape over its Ethernet port). You'll need a mechanism for moving files to and from the code-signing machine. A USB-connected disk or CD-RW will work, or if your files are small you can use a floppy disk.
- Install crypt_prog on the code signing machine (it's easiest if the machine runs Unix/Linux; Windows can be used but requires Visual Studio 2003).
- Run 'crypt_prog -genkey' to create a code-signing key pair. Copy the public key to your server. Keep the private key on the code-signing machine, make a permanent, secure copy of the key pair (e.g. on a CD-ROM that you keep locked up), and delete all other copies of the private key.
- To sign an executable file, move it to the code-signing machine, run 'crypt_prog -sign' to produce the signature file, then move the signature file to your server.
- Use update_versions to install your application, including its signature files, in the download directory and database.

There are less-secure variants; e.g. you could keep the private key on a CD-ROM that is only mounted during signature generation, on a machine that is disconnected during signature generation. But we do not recommend this; a hacked computer could be running a hidden program that steals the private key and transmits it when the computer is connected again.

42. Upgrading server software

The BOINC server software (scheduler, daemons, web pages) is continually improved and debugged. We recommend that projects upgrade to the latest version every few weeks or so. There may also be points upgrades are mandatory to continue working with current client software.

The steps in upgrading are as follows:

1. (Optional) stop the project, and make backups of the project database and the project tree.
2. Download (using CVS) the current source code. Compile it in your BOINC source directory.
3. Run the `upgrade` script:

   ```
   cd source/tools
   upgrade project_name
   ```

   The `upgrade` script copies files from the source/html/, source/sched and source/tool directories to the corresponding project directories (the default project root directory is $HOME/projects/project_name; `upgrade` takes an optional environment variables INSTALL_DIR specifying the project's root directory).
4. Update your project's database if needed:

   ```
   cd project/html/ops
   ```

   and look at the file db_update.php. This has a number of functions with names like

   ```
   update_8_05_2005()
   ```

   Each function performs a particular database update. You must perform all updates, in sequence, since your last server software upgrade. (If you're not sure when that was, you can use mysql to see that current format of your database, e.g., to see the fields in the 'user' table, type

   ```
   mysql project_name
   > explain user;
   ```

   To do a particular update, edit db_update.php so that (at the bottom) it calls that function. Then do

   ```
   php db_update.php
   ```

   Repeat this for the necessary updates, in increasing chronological order.
5. Start the project, and check log files to make sure everything is OK. Run the BOINC client and test basic functions (attaching to project, getting work).

43. Multiple server hosts

When you initially create a BOINC project using make_project, everything runs on a single host: web server, scheduling server, daemons, tasks, database server, file upload handler.

You can increase the capacity of your server by spreading these tasks across different hosts. The following rules apply:

- The hosts should share a common network file system, and path of the project directory, on any host, should map to the same place. (Exception: the database server can run anywhere).
- The project admin account should exist on all hosts, and that user should be able to use 'ssh' to run commands on any other host without typing a password.
- One host is designated as the project's **main host** in config.xml. **The 'start', 'stop', and 'status' scripts should normally be run on the main host** (if you run them on a different host X, they'll affect only daemons and tasks on host X).
- The project admin account on all hosts must be able to access the project's MySQL database. (Exception: data servers and file upload handlers don't need DB access).

Host locations are specified as follows:

- Scheduling servers are listed in the project's master page.
- The hosts on which tasks and daemons are run are specified in the config.xml file.
- Data servers are listed in workunit template files.
- File upload handlers are listed in result template files.
- Your web server runs on the host to which your project URL is mapped.

44. Beta-test applications

It's important to test new applications on a wide range of hosts, since bugs may appear only with particular OS versions, memory sizes, display types, usage patterns, and so on. It's handy to use volunteers to do this testing, since they provide the needed diversity of hosts.

One way to implement this is to create a separate test project. This has two disadvantages:

- There is overhead in creating and maintaining a separate project.
- The credit accrued by testers goes to a different project.

BOINC provides a way to do beta testing in the context of your existing project. You can let users volunteer to run test applications, warning them in advance that these applications are more likely to crash. These users will get a mixture of regular and test results, and they'll get credit for both. Here's how to do it:

- Upgrade to the BOINC server software of Oct 25 2006 or later.
- Create a new application using xadd. Include <beta>1</beta> in the <app> element to designate it as a beta-test application.
- Add a validator and assimilator for the test application.
- Include the line

  ```
  $project_has_beta = true;
  ```

  in your html/project/project_specific_prefs.inc file. This will add a 'Run test applications?' option to your project-specific preferences.
- Publicize this on your web site and wait for some users to set their preferences to allow test apps. (Note: this flag is stored in XML in the project_prefs field of the user table; scan for <allow_beta_work>1</allow_beta_work>).
- Create application versions for the test application.
- Create work (as needed for testing) for the test application. To prevent test work from dominating regular work, either use the -allapps feeder option (and give the test app a small weight) or make a work generator for the test app that maintains only a small number of unsent results.

45. Work-handling daemons

A BOINC project includes of a set of daemons for generating and handling work. Each program should be listed as a daemon in the config.xml file. Most daemons have the command-line options:

-d N	Sets the verbosity level. 1 = critical messages only, 2 = normal messages, 3 = detailed debugging info.
-one_pass	Process all available items, then quit.
-mod n i	Handle only workunits for which mod(id, n) = i. This lets you run the daemon on arbitrarily many machines. (available for feeder, transitioner, validator).
-sleep_interval N	How long to sleep when there's nothing to do.

Work generator

There is one work generator per application. It creates workunits and the corresponding input files. It is application-specific, and uses BOINC library functions for registering the workunits in the database.

During testing, you can create a single workunit using create_work, then use the daemon program make_work to copy this workunit as needed to maintain a given supply of work.

Feeder

This program is supplied by BOINC and is application independent. It creates a shared-memory segment used to pass database records to CGI scheduler processes. This data includes applications, application versions, and 'work items' (an unsent result and its corresponding workunit). It has the following command-line options:

-random_order	Enumerate work items in order of increasing result.random
-priority_order	Enumerate work items in order of decreasing result.priority
-priority_order_create_time	Enumerate work items in order of decreasing result.priority, then increasing workunit.create_time
-sleep_interval N	Sleep N seconds if nothing to do
-allapps	Interleave work items from all applications. Weight applications according to the value of their 'weight' field; if all weights are zero, results are interleaved uniformly. Without this option, runnable results are enumerated in an indeterminate order, and there may be periods when only results from one application are available for sending.
-purge_stale X	remove work items from the shared memory segment that have been there for longer then x minutes but haven't been assigned

If a user's project preferences include elements of the form <app_id>N</app_id> then the scheduler will send the user work only from those applications.

Transitioner

This program is supplied by BOINC and is application independent. It handles state transitions of workunits and results. It generates initial results for workunits, and generates more results when timeouts or errors occur.

Validator

There is one validator per application. It compares redundant results and selects a **canonical result** representing the correct output, and a **canonical credit** granted to users and hosts that return the correct output. Depending on your application, you may be able to use a BOINC-supplied validator, or you may have to develop a customized validator; details are here.

Assimilator

There is one assimilator per application. It handles workunits that are 'completed': that is, that have a canonical result or for which an error condition has occurred. Handling a successfully completed result might involve record results in a database and perhaps generating more work.

File deletion

The application-independent program file_deleter deletes input and output files when they are no longer needed.

Database purging

The application-independent program db_purge removes work-related database entries when they are no longer needed. This keeps your database at a constant size even when your project runs for a long time.

46. Generating work

As described earlier, a workunit represents the inputs to a computation. The steps in creating a workunit are:

- Write XML 'template files' that describe the workunit and its corresponding results. Generally the same templates will be used for a large number work of workunits.
- Create the workunit's input file(s) and place them in the download directory.
- Invoke a BOINC function or script that creates a database record for the workunit.

Once this is done, BOINC takes over: it creates one or more results for the workunit, distributes them to client hosts, collects the output files, finds a canonical result, assimilates the canonical result, and deletes files.

During the testing phase of a project, you can use the make_work daemon to replicate a given workunit as needed to maintain a constant supply of work. This is useful while testing and debugging the application.

Workunit and result template files

A workunit template file has the form

```
<file_info>
    <number>0</number>
    [ <sticky/>, other attributes]
</file_info>
[ ... ]
<workunit>
    <file_ref>
        <file_number>0</file_number>
        <open_name>NAME</open_name>
    </file_ref>
    [ ... ]
    [ <command_line>-flags xyz</command_line> ]
    [ <rsc_fpops_est>x</rsc_fpops_est> ]
    [ <rsc_fpops_bound>x</rsc_fpops_bound> ]
    [ <rsc_memory_bound>x</rsc_memory_bound> ]
    [ <rsc_disk_bound>x</rsc_disk_bound> ]
    [ <delay_bound>x</delay_bound> ]
    [ <min_quorum>x</min_quorum> ]
    [ <target_nresults>x</target_nresults> ]
    [ <max_error_results>x</max_error_results> ]
    [ <max_total_results>x</max_total_results> ]
    [ <max_success_results>x</max_success_results> ]
    [ <credit>X</credit> ]
</workunit>
```

The components are:

<file_info>, <file_ref>	Each pair describes an input file and the way it's referenced.
<command_line>	The command-line arguments to be passed to the main program.
<credit>	The amount of credit to be granted for successful completion of this workunit. Use this only if you know in advance how many FLOPs it will take. Your validator must use get_credit_from_wu() as its compute_granted_credit() function.
Other elements	Work unit attributes

Workunit database records include a field, 'xml_doc', that is an XML-format description of the workunit's input files. This is derived from the workunit template as follows:

- Within a <file_info> element, <number>x</number> identifies the order of the file. It is replaced with elements giving the filename, download URL, MD5 checksum, and size.

- Within a <file_ref> element, <file_number>x</file_number> is replaced with an element giving the filename.

A result template file has the form

```
<file_info>
    <name><OUTFILE_0/></name>
    <generated_locally/>
    <upload_when_present/>
    <max_nbytes>32768</max_nbytes>
    <url><UPLOAD_URL/></url>
</file_info>
<result>
    <file_ref>
        <file_name><OUTFILE_0/></file_name>
        <open_name>result.sah</open_name>
    </file_ref>
</result>
```

Result database records include a field, 'xml_doc_in', that is an XML-format description of the result's output files. This is derived from the result template as follows:

- <OUTFILE_n> is replaced with a string of the form 'wuname_resultnum_n' where wuname is the workunit name and resultnum is the ordinal number of the result (0, 1, ...).
- <UPLOAD_URL/> is replaced with the upload URL.

Moving input files to the download directory

If you're using a flat download directory, just put input files in that directory. If you're using hierarchical upload/download directories, you must put each input file in the appropriate directory; the directory is determined by the file's name. To find this directory, call the C++ function

```
dir_hier_path(
    const char* filename,
    const char* root,       // root of download directory
    int fanout,             // from config.xml
    char* result,           // path of file in hierarchy
    bool create_dir=false   // create dir if it's not there
);
```

If you're using scripts, you can invoke the program

```
dir_hier_path filename
```

It prints the full pathname and creates the directory if needed. Run this in the project's root directory. For example:

```
cp test_workunits/12ja04aa `bin/dir_hier_path 12ja04aa`
```

copies an input file from the test_workunits directory to the download directory hierarchy.

Creating workunit records

Workunits can be created using either a script (using the `create_work` program) or a program (using the `create_work()` function). The input files must already be in the download hierarchy.

The utility program is

```
create_work
    -appname name                       // application name
    -wu_name name                       // workunit name
    -wu_template filename               // WU template filename
        // relative to project root; usually in templates/
    -result_template filename           // result template filename
        // relative to project root; usually in templates/
    [ -batch n ]
    [ -priority n ]

    // The following may be passed in the WU template,
    // or as command-line arguments to create_work,
    // or not passed at all (defaults will be used)

    [ -command_line "-flags foo" ]
```

```
    [ -rsc_fpops_est x ]
    [ -rsc_fpops_bound x ]
    [ -rsc_memory_bound x ]
    [ -rsc_disk_bound x ]
    [ -delay_bound x ]
    [ -min_quorum x ]
    [ -target_nresults x ]
    [ -max_error_results x ]
    [ -max_total_results x ]
    [ -max_success_results x ]
    [ -additional_xml 'x' ]

    infile_1 ... infile_m           // input files
```

The program must be run in the project root directory. The workunit parameters are documented here. The -additional_xml argument can be used to supply, for example, <credit>12.4</credit>.

BOINC's library (backend_lib.C,h) provides the functions:

```
int create_work(
    DB_WORKUNIT&,
    const char* wu_template,                  // contents, not path
    const char* result_template_filename,     // relative to project root
    const char* result_template_filepath,     // absolute or relative to current dir
    const char** infiles,                     // array of input file names
    int ninfiles
    SCHED_CONFIG&,
    const char* command_line = NULL,
    const char* additional_xml = NULL
);
```

create_work() creates a workunit. The arguments are similar to those of the utility program; some of the information is passed in the DB_WORKUNIT structure, namely the following fields:

```
name
appid
```

The following may be passed either in the DB_WORKUNIT structure or in the workunit template file:

```
rsc_fpops_est
rsc_fpops_bound
rsc_memory_bound
rsc_disk_bound
batch
delay_bound
min_quorum
target_nresults
max_error_results
max_total_results
max_success_results
```

Examples

Making one workunit

Here's a program that generates one workunit (error-checking is omitted for clarity):

```
#include "backend_lib.h"

main() {
    DB_APP app;
    DB_WORKUNIT wu;
    char wu_template[LARGE_BLOB_SIZE];
    char* infiles[] = {"infile"};

    SCHED_CONFIG config;
    config.parse_file();

    boinc_db.open(config.db_name, config.db_host, config.db_passwd);
    app.lookup("where name='myappname'");

    wu.clear();     // zeroes all fields
    wu.appid = app.id;
    wu.min_quorum = 2;
    wu.target_nresults = 2;
    wu.max_error_results = 5;
    wu.max_total_results = 5;
    wu.max_success_results = 5;
    wu.rsc_fpops_est = 1e10;
```

```
    wu.rsc_fpops_bound = 1e11;
    wu.rsc_memory_bound = 1e8;
    wu.rsc_disk_bound = 1e8;
    wu.delay_bound = 7*86400;
    read_filename("templates/wu_template.xml", wu_template, sizeof(wu_template));
    create_work(
        wu,
        wu_template,
        "templates/results_template.xml",
        "templates/results_template.xml",
        infiles,
        1,
        config
    );
}
```

This program must be run in the project directory since it expects to find the config.xml file in the current directory.

Making lots of workunits

If you're making lots of workunits (e.g. to do the various parts of a parallel computation) you'll want the workunits to differ either in their input files, their command-line arguments, or both.

For example, let's say you want to run a program on ten input files 'file0', 'file1', ..., 'file9'. You might modify the above program with the following code:

```
    char filename[256];
    char* infiles[1];
    infiles[0] = filename;
    ...
    for (i=0; i<10; i++) {
        sprintf(filename, "file%d", i);
        create_work(
            wu,
            wu_template,
            "templates/results_template.xml",
            "templates/results_template.xml",
            infiles,
            1,
            config
        );
    }
```

Note that you only need one workunit template file and one result template file.

Now suppose you want to run a program against a single input file, but with ten command lines, '-flag 0', '-flag 1', ..., '-flag 9'. You might modify the above program with the following code:

```
    char command_line[256];
    ...
    for (i=0; i<10; i++) {
        sprintf(command_line, "-flag %d", i);
        create_work(
            wu,
            wu_template,
            "templates/results_template.xml",
            "templates/results_template.xml",
            infiles,
            1,
            config,
            command_line
        );
    }
```

Again, you only need one workunit template file and one result template file.

47. Validation

In BOINC, the process of **validation** does two things:

- it compares redundant results and decides which ones are to be considered correct;
- it decides how much credit to grant to each correct result.

A **validator** is a daemon program. You must supply a validator for each application in your project, and include it in the <daemons> section of your project configuration file.

Depending on various factors, you may be able to use a standard validator that comes with BOINC, or you may have to develop a custom validator.

- If your application generates exactly matching results (either because it does no floating-point arithmetic, or because you use homogeneous redundancy) then you can use the 'sample bitwise validator' (see below).
- If you are using BOINC for 'desktop grid' computing (i.e. you trust all the participating hosts) then you can use the 'sample trivial validator' (see below).
- Otherwise, you'll need to develop a custom validator for your application. BOINC supplies a simple validator framework in which you plug in a few short application-specific functions. This is sufficient for most projects. If you need more control over the validation process, you can use BOINC's advanced validator framework.

Two standard validators are supplied:

- The **sample_bitwise_validator** requires a strict majority, and regards results as equivalent only if they agree byte for byte.
- The **sample_trivial_validator** regards any two results as equivalent if their CPU time exceeds a given minimum.

Command-line arguments

A validator (either standard or custom) has the following command-line arguments:

-app appname	Name of the application
[-one_pass_N_WU N]	Validate at most N WUs, then exit
[-one_pass]	Make one pass through WU table, then exit
[-mod n i]	Process only WUs with (id mod n) == i. This option lets you run multiple instances of the validator for increased performance.
[-max_claimed_credit X]	If a result claims more credit than this, mark it as invalid.
[-max_granted_credit X]	Grant no more than this amount of credit to a result.
[-grant_claimed_credit]	If set, grant the claimed credit, regardless of what other results for this workunit claimed. These is useful for projects where different instances of the same job can do much different amounts of work.

48. Result assimilation

Projects must create one assimilator program per application. This is done by linking the program **sched/assimilate.C** with an application-specific function of the form

```
int assimilate_handler(
    WORKUNIT& wu, vector<RESULT>& results, RESULT& canononical_result
);
```

This is called when either

- The workunit has a nonzero error mask (indicating, for example, too many error results). In this case the handler might write a message to a log or send an email to the application developer.
- The workunit has a canonical result. In this case wu.canonical_resultid will be nonzero, canonical_result will contain the canonical result. Your handler might, for example, parse the canonical result's output file and write its contents to a separate database.

In both cases the 'results' vector will be populated with all the workunit's results (including unsuccessful and unsent ones). All files (both input and output) will generally be on disk.

It's possible that both conditions might hold.

If assimilate_handler() returns zero, the workunit record will be marked as assimilated. If assimilate_handler() returns nonzero, the assimilator will log an error message and exit. Typically the function should return nonzero for any error condition. This way the system administrator can fix the problem before any completed or erroneous workunits are mis-handled by BOINC.

You can use BOINC's back-end utility functions to get file pathnames and open files.

49. Server-side file deletion

Files are deleted from the data server's upload and download directories by the **file_deleter** daemon. Typically you don't need to customize this. The default file deletion policy is:

- A workunit's input files are deleted when all results are 'over' (reported or timed out) and the workunit is assimilated.
- A result's output files are deleted after the workunit is assimilated. The canonical result is handled differently, since its output files may be needed to validate results that are reported after assimilation; hence its files are deleted only when all results are over, and all successful results have been validated.

Command-line options are:

-preserve_wu_files	Don't delete input files
-preserve_result_files	Don't delete output files
-retry_errors	Retry file deletions that failed previously.

In some cases you may not want files to be deleted. There are three ways to accomplish this:

- Run the file_deleter daemon with the -preserve_wu_files and/or the -preserve_result_files command-line options.
- Include <no_delete/> in the <file_info> element for a file in a workunit or result template. This lets you suppress deletion on a file-by-file basis.
- Include `nodelete` in the workunit name.

You may need to implement your own scheme for deleting files, to avoid overflowing data server storage.

50. Database purging utility

As a BOINC project operates, the size of its workunit and result tables increases. Eventually they become so large that adding a field or building an index may take hours or days.

To address this problem, BOINC provides a utility **db_purge** that writes result and WU records to XML-format archive files, then deletes them from the database.

Workunits are purged only when their input files have been deleted. Because of BOINC's file-deletion policy, this implies that all results are completed. So when a workunit is purged, all its results are purged too.

Run db_purge from the project's bin/ directory. It will create an archive/ directory and store archive files there.

db_purge is normally run as a daemon, specified in the config.xml file. It has the following command-line options:

-min_age_days N	Purge only WUs with mod_time at least N days in the past. Recommended value: 7 or so. This lets users examine their recent results.
-max N	Purge at most N WUs, then exit
-max_wu_per_file N	Write at most N WUs to each archive file. Recommended value: 10,000 or so.
-zip	Compress archive files using zip
-gzip	Compress archive files using gzip
-d N	Set logging verbosity to N (1,2,3)

Archive file format

The archive files have names of the form wu_archive_TIME and result_archive_TIME where TIME is the Unix time the file was created. In addition, db_purge generates index files 'wu_index' and 'result_index' associating each WU and result ID with the timestamp of its archive file.

The format of both type of index files is a number of rows each containing:

```
ID     TIME
```

The ID field of the WU or result, 5 spaces, and the timestamp part of the archive filename where the record with that ID can be found.

The format of a record in the result archive file is:

```
<result_archive>
    <id>%d</id>
  <create_time>%d</create_time>
  <workunitid>%d</workunitid>
  <server_state>%d</server_state>
  <outcome>%d</outcome>
  <client_state>%d</client_state>
  <hostid>%d</hostid>
  <userid>%d</userid>
  <report_deadline>%d</report_deadline>
  <sent_time>%d</sent_time>
  <received_time>%d</received_time>
  <name>%s</name>
  <cpu_time>%.15e</cpu_time>
  <xml_doc_in>%s</xml_doc_in>
  <xml_doc_out>%s</xml_doc_out>
  <stderr_out>%s</stderr_out>
  <batch>%d</batch>
  <file_delete_state>%d</file_delete_state>
  <validate_state>%d</validate_state>
  <claimed_credit>%.15e</claimed_credit>
```

```
  <granted_credit>%.15e</granted_credit>
  <opaque>%f</opaque>
  <random>%d</random>
  <app_version_num>%d</app_version_num>
  <appid>%d</appid>
  <exit_status>%d</exit_status>
  <teamid>%d</teamid>
  <priority>%d</priority>
  <mod_time>%s</mod_time>
</result_archive>
```

The format of a record in the WU archive file is:

```
<workunit_archive>
    <id>%d</id>
  <create_time>%d</create_time>
  <appid>%d</appid>
  <name>%s</name>
  <xml_doc>%s</xml_doc>
  <batch>%d</batch>
  <rsc_fpops_est>%.15e</rsc_fpops_est>
  <rsc_fpops_bound>%.15e</rsc_fpops_bound>
  <rsc_memory_bound>%.15e</rsc_memory_bound>
  <rsc_disk_bound>%.15e</rsc_disk_bound>
  <need_validate>%d</need_validate>
  <canonical_resultid>%d</canonical_resultid>
  <canonical_credit>%.15e</canonical_credit>
  <transition_time>%d</transition_time>
  <delay_bound>%d</delay_bound>
  <error_mask>%d</error_mask>
  <file_delete_state>%d</file_delete_state>
  <assimilate_state>%d</assimilate_state>
  <hr_class>%d</hr_class>
  <opaque>%f</opaque>
  <min_quorum>%d</min_quorum>
  <target_nresults>%d</target_nresults>
  <max_error_results>%d</max_error_results>
  <max_total_results>%d</max_total_results>
  <max_success_results>%d</max_success_results>
  <result_template_file>%s</result_template_file>
  <priority>%d</priority>
  <mod_time>%s</mod_time>
</workunit_archive>
```

51. Administrative web interface

BOINC's administrative web interface provides interfaces for

- Browsing the database
- Screening user profiles
- Viewing recent results
- Browsing stripcharts
- Browsing log files
- Creating user accounts

52. Debugging server components

A grab-bag of techniques for debugging BOINC server software:

Log files

Most error conditions are reported in the log files. Make sure you know where these are. If you're interested in the history of a particular WU or result, grep for WU#12345 or RESULT#12345 (12345 represents the ID) in the log files. The html/ops pages also provide an interface for this.

Database query tracing

If you uncomment the symbol SHOW_QUERIES in db/db_base.C, and recompile everything, all database queries will be written to stderr (for daemons, this goes to log files; for command-line apps it's written to your terminal). This is verbose but extremely useful for tracking down database-level problems.

Getting core dumps from the scheduler

In sched/main.C put:

```
#define DUMP_CORE_ON_SEGV 1
```

and recompile.

Running the scheduler under a debugger

The scheduler is a CGI program. It reads from stdin and writes to stdout, so you can also run it with a command-line debugger like gdb:

- Copy the "scheduler_request_X.xml" file from a client to the machine running the scheduler. (X = your project URL)
- Run the scheduler under the debugger, giving it this file as stdin, i.e.:

```
gdb cgi
(set a breakpoint)
r < scheduler_request_X.xml
```

- You may have to doctor the database as follows:

```
update host set rpc_seqno=0, rpc_time=0 where hostid=N
```

  to keep the scheduler from rejecting the request.

This is useful for figuring out why your project is generating 'no work available' messages. As an alternative to this, edit handle_request.C, and put a call to debug_sched(sreq, sreply, "../debug_sched") just before sreply.write(fout). Then, after recompiling, touch a file called 'debug_sched' in the project root directory. This will cause transcripts of all subsequent scheduler requests and replies to be written to the cgi-bin/ directory with separate small files for each request. The file names are sched_request_H_R where H=hostid and R=rpc sequence number. This can be turned off by deleting the 'debug_sched' file.

MySQL interfaces

You should become familiar with MySQL tools such as

- mytop: like 'top' for MySQL
- the mysql interpreter ('mysql') and in particular the 'show processlist;' query.
- MySQLadmin: general-purpose web interface to MySQL

53. Log rotation

The log files generated by BOINC's daemons and servers can grow to gigabyte size in a few days or weeks, and will eventually fill up your disk. When this happens your project will grind to a halt.

In addition, if you use db_dump to export statistics, directories with names of the form **html/stats_2006_4_3_15_50_2** will build up, and you'll need to delete and possibly archive them. Most projects do this manually.

Initially you can deal with log files by hand, but eventually you may want an automated solution. Log rotation typically must be done while the project is stopped.

(Note: Eric Myers wrote a detailed page about log rotation, which is here).

Here's a shell script (from Nicolas Alvarez) that tars and compresses log files (you'll still need to eventually deal with the compressed files). You can run this script as a periodic task in config.xml.

```
#!/bin/bash
cd ..
./bin/stop
pushd ./log_servername
BACKUP_DIR=`date --utc +backup_%Y_%m_%d`
mkdir $BACKUP_DIR
mv *.log *.out $BACKUP_DIR
pushd ..
./bin/start&
popd
tar cjvf $BACKUP_DIR.tar.bz2 $BACKUP_DIR
rm -rf $BACKUP_DIR
popd
```

Other projects the Linux 'logrotate' program. For example, Predictor@home uses the following logrotate input file (they run this while the project is down for database backups):

```
/export/projects/predictor/log_predictor1/cgi.log{
    compress
    dateext
    maxage 365
    rotate 99
    size=+1096k
    notifempty
    missingok
    create 664 wwwrun users
    postrotate
        /etc/init.d/apache2 reload
    endscript
}

/export/projects/predictor/log_predictor1/file_upload_handler.log{
    compress
    dateext
    maxage 365
    rotate 99
    size=+1096k
    notifempty
    missingok
    create 664 wwwrun users
    postrotate
        /etc/init.d/apache2 reload
    endscript
}

/export/projects/predictor/log_predictor1/transitioner.log{
    compress
    dateext
    maxage 365
    rotate 99
    size=+1096k
    notifempty
    missingok
    create 644 boinc users
}

/export/projects/predictor/log_predictor1/taskmaster.log{
```

54. Retrieving file lists

To instruct a host to send a list of all persistent files, use the function

```
request_file_list(int host_id)
```

or the command line program (run from the project root directory)

```
request_file_list [ –host_id X ]
```

If -host_id is absent, get file lists for all hosts.

A message is created for the specific host (or all hosts) and added to the msg_to_host table in the database. The upload message included in the next RPC reply to the host.

The file list will be included in the next scheduler RPC request. You must modify the scheduler to parse and store it.

55. Retrieving files

A persistent file can be retrieved from a specific host. This can be done using the function

```
get_file(int host_id, const char* file_name)
```

or the command line program (run in the project root dir)

```
get_file –host_id X –file_name Y
```

This program must be run in the project's root directory.

get_file() creates a result with a name of the form:

```
get_FILENAME_HOSTID_TIME
```

Example: get_test.mpg_34_123456789 is a result representing the upload of test.mpg from host number 34 at time 1234567891.

An upload message is created for the specific host and added to the msg_to_host table in the database. This message instructs the client to upload the file and acknowledge a successful upload. The upload message included in the next RPC reply to the host. The message has the form:

```
<app>
    <name>FILE_MOVER</name>
</app>
<app_version>
    <app_name>FILE_MOVER</app_name>
    <version_num>BOINC_MAJOR_VERSION</version_num>
</app_version>
<file_info>
    <name>file_name</name>
    <url>upload_dir</url>
    <max_nbytes>1e10</max_nbytes>
    <upload_when_present/>
</file_info>
    RESULT_XML
<workunit>
    <name>result.name</name>
    <app_name>FILE_MOVER</app_name>
</workunit>
```

Include

```
<msg_to_host/>
```

in config.xml. Currently

```
<ignore_upload_certificates/>
```

needs to be included as there is no way to send upload certificates with these files.

56. Sending files

To send a file to a specific host, use the function

```
send_file(int host_id, const char* file_name)
```

or the command line program (run from the project root directory)

```
send_file –host_id X –file_name Y
```

send_file creates a result and initializes it with a name of the form send_FILENAME_HOSTID_TIME.

A message is created for the host and added to the msg_to_host table. This message instructs the client to download the file and acknowledge a successful download. The download message included in the next RPC reply to the host. The message has the form:

```
<app>
        <name>FILE_MOVER</name>
</app>
<app_version>
        <app_name>FILE_MOVER</app_name>
        <version_num>n</version_num>
</app_version>
<result>
    <wu_name>x</wu_name>
    <name>y</name>
</result>
<file_info>
        <name>file_name</name>
        <url>download_dir/file_name</url>
        <md5_cksum>md5</md5_cksum>
        <nbytes>file->nbytes</nbytes>
        <sticky/>
</file_info>
<workunit>
        <name>result.name</name>
        <app_name>FILE_MOVER</app_name>
        <file_ref>
                <file_name>file_name</file_name>
        </file_ref>
</workunit>
```

57. Deleting files

To delete a file from a host, use the function

```
delete_file(int host_id, const char* file_name)
```

or the command line program (run from the project root dir)

```
delete_file –host_id X –file_name Y
```

delete_file() creates a message for the specific host and adds it to the msg_to_host table. This message instructs the client to delete the file. The message is included in the next scheduler reply to the host. The message XML has the form

```
<delete_file_info>file_name</delete_file_info>
```

58. Web site overview

Customizing the default web site

When you create a BOINC project using make_project, a web site is created for you. This consists of a front page (html/user/index.php), which links to pages where users can log in, edit preferences, create profiles, and so on.

Before your project goes public, you'll want to change this web site by adding content specific to your project, and by giving the web site a graphical identity specific to your project. Make sure you do a good job; your web site has a large impact on your project's ability to recruit and retain participants.

Some of this customization can be done by editing the main page (index.php) and the stylesheet (white.css). Other aspects are changed using a configuration file, described below.

Web configuration file

The file 'html/project/project.inc' serves as a configuration file for your web site. It exists in a separate directory (html/project) so that you can put this directory under CVS and put all project-specific web files there (create symbolic links from html/inc and html/user).

project.inc is generated by make_project with default values. It includes constants:

PROJECT	The name of your project
MASTER_URL	Your project's master URL
URL_BASE	Base URL for web pages (usually same as master URL)
STYLESHEET	Name of stylesheet file
COPYRIGHT_HOLDER	Name of copyright holder
SYS_ADMIN_EMAIL	Users are directed here if they have complaints about message-board moderation. Also, messages about user-of-the-day running low are sent here.
FORUM_MODERATION_EMAIL_USER_ID	Moderation-related emails (such as user complaints) are sent here.
INVITE_CODES	regular expression used for controlling account creation.
EMAIL_FROM	'from' address for emails
EMAIL_FROM_NAME	'from' name for emails

and functions:

project_banner()	prints page header
project_banner()	prints page footer
show_profile_heading1(), show_profile_heading2()	text on user profile page
show_profile_question1(), show_profile_question2()	text on user profile page
project_workunit()	prints project-specific text on workunit page
project_user_summary()	prints project-specific text on user page
project_user_page_private()	prints project-specific text on private user page

and variables:

USE_PHPMAILER	Set to true if you use PHPMailer. In this case you must download PHPMailer and put it (i.e. the directory 'phpmailer') in your html/inc directory.

PHPMAILER_HOST	The Host argument to PHPMailer; typically a semicolon-separated list of SMTP servers.
PHPMAIL_MAILER	The Mailer argument to PHPMailer; typically 'sendmail', 'mail', or 'smtp'.

59. Message boards

BOINC provides its own software for web based message-boards (also called 'forums'). Messages, and descriptions of the message boards, are stored in the BOINC database. Two types of message boards are supported:

- **Questions and answers**: These provide a 'self-organizing FAQ' that shows questions (sorted so that current and frequently-asked questions are listed first) and answers (sorted so that 'good' answers are listed first).
- **Forums**: These are conventional message boards, typically sorted so that most recent messages are first.

To create message boards for your project, edit the file `html/ops/create_forums.php` to create a set of message boards appropriate for your project, and then run that script, i.e. type

```
php create_forums.php
```

User attributes

The Special user feature allows certain users, like project administrators, developers etc., to be shown with that title under their name in the forums. It is important that people who are new to a project knows who to pay attention to - and this is a way of giving them a hint. To enable the feature simply run a query on the forum_preferences table. You can currently use the following list of titles:

```
$special_user_bitfield[0]="Forum moderator";
$special_user_bitfield[1]="Project administrator";
$special_user_bitfield[2]="Project developer";
$special_user_bitfield[3]="Project tester";
$special_user_bitfield[4]="Volunteer developer";
$special_user_bitfield[5]="Volunteer tester";
$special_user_bitfield[6]="Project scientist";
```

So if the project administrator has the user number 42 run this query to make him a moderator and project administrator:

```
UPDATE forum_preferences SET special_user=1100000 where userid=42;
```

Moderation

Post-level moderation abilities

- When browsing posts a moderator is able to delete (or hide really) a single post by clicking the 'delete' link in the top information bar for that post. It is possible to select a reason why the that post was deleted (this will be displayed to the poster). Also a message can be sent directly to the poster's mail address by typing in additional text in a specific field when deleting.
- If a post was deleted by accident it will still be available to moderators (it now has 'deleted post' in red in the top information bar for that post). A moderator can click 'undelete' to make the post available to the public again.
- Moderators can move a single post to another thread by pressing the top information bar 'move post' link. The moderator will need to know the destination thread ID. Again a message can be sent to the poster.
- TODO: Moderators can set the post rating (do we want this?)

Thread-level moderation abilities

- A moderator can delete an entire thread using the 'delete thread' link located at the very top of the page.
- TODO: move threads to other forum
- TODO: undelete threads
- Users can vote negatively against the first post to eventually mark the thread as non-interesting (is this known?)

User-level moderation abilities

- Users can vote negatively against posts to eventually hide them.
- users can ignore other users.
- It is possible to list all posts for a user and look at them one by one.

Word-level moderation/filtering abilities

None, all words are allowed.

Sub-post-level moderation abilities

None, moderators are not allowed to edit people's posts.

60. Web and GUI translations

BOINC has a mechanism for non-English translations of

- Parts of this site (the Download and BOINC user survey pages, and related pages)
- The BOINC Manager
- Parts of the BOINC-supplied portion of project web sites
- The project-specific parts of project web sites

Instructions for volunteer translators

Translations are done by volunteers. If you're interested in helping:

- Email the translation manager. For BOINC this is **translate at boinc.berkeley.edu**. Use this address also for SETI@home translations. For other projects, contact the project.
- Obtain (typically via CVS) the 'authoritative' translation file. Usually this is en.po (English).
- Create a translation file for your language. You can do this using a text editor or a specialized tool such as poedit.
- Send this to the translation manager, who will then install it on the project's web site. Check all relevant pages and fix as needed.
- Subscribe to the boinc_loc at ssl.berkeley.edu email list, which is for translation-related discussion and announcements.
- Because web sites are dynamic, you will have to periodically update your translation. You can do this efficiently by looking at the CVS diffs of the authoritative translation file.

Translation files

All translations are based on **translation files**. Translation files are in PO format, which is described here. These have names like 'da.po' (Danish) and 'en.po' (English). It's very simple. For example:

en.po (English)

```
msgid "APPS_VERSION"
msgstr "Current version"

msgid "APPS_DESCRIPTION"
msgstr "$PROJECT currently has the following applications. "
"When you participate in $PROJECT, work for one or more "
```

```
"of these applications will be assigned to your computer. "
"The current version of the application will be downloaded "
"to your computer. This happens automatically; you don't have to do anything. "
```

da.po (Danish)

```
msgid "APPS_VERSION"
msgstr "Nuværende version"

msgid "APPS_DESCRIPTION"
msgstr "$PROJECT har i øjeblikket følgende applikationer. "
"Når du deltager i $PROJECT vil arbejde fra en eller flere "
"af disse applikationer blive tildelt din computer. "
"Den nuværende version af applikationen vil blive downloadet "
"til din computer. Dette sker automatisk - du behøver ikke at gøre noget."
```

Each translation has a token CHARSET whose value should be the character set (e.g. iso-8859-1) used in the translation.

Here are links to the translation files for

- this web site
- the BOINC-supplied part of project web sites.
- the BOINC Manager.

Translatable web pages

Translatable web pages must be PHP, and must include

```
require_once("../inc/translation.inc");
```

Literal text is replaced with `tr(TOKEN)`, where TOKEN is a short string representing the text. For example,

```
page_head("Current version");
```

is replaced with

```
page_head(tr(APPS_VERSION));
```

For BOINC projects, the directory html/user/translations contains a number of 'translation files'. When a person accesses the page, the browser passes a list of languages. The BOINC PHP code finds the first of these languages for which a translation is available, or English if none. As the page is generated, each call to `tr()` replaces the token with the corresponding translated text.

In developing web pages, keep in mind that word order differs between languages, so you should avoid breaking a sentence up into multiple translation units. For example, use constructs like

```
msgid "ACTIVATE_OR_CREATE"
msgstr "Already have an original 'Classic' account as of May 14, 2004? "
    "<br>We've transferred it, just %sactivate it%s. "
    "%sOtherwise %screate a new account%s."
```

with the corresponding PHP:

```
printf(tr(ACTIVATE_OR_CREATE),
    "<a href=sah_email_form.php>", "</a>",
    "<p><img border=0 src=images/arrow_right.gif width=9 height=7>",
    "<a href=create_account_form.php>", "</a>"
);
```

Project-specific translations

The web site of a BOINC-based project involves both

- BOINC pages, such as the forms for creating accounts. These are part of the BOINC source code distribution, and are updated periodically by BOINC.
- Project-specific pages (and BOINC pages that are modified by the project).

To allow translations of both types of pages, a project can haves its own 'project-specific translation files'. These are stored in a directory html/user/project_specific_translations. Project-specific translation files override BOINC translation files.

BOINC Manager translations

Menu names and other text in the BOINC manager are stored in files in **boinc/locale/client/**.

61. Server status

Each project should export its server status in two forms:

- **URL/server_status.php**: human-readable (web page).
- **URL/server_status.php?xml=1**: XML format, as follows:

```
<server_status>
    <update_time>1127772607</update_time>
        <daemon_status>
        <daemon>
            <host>jocelyn</host>
            <command>BOINC database</command>
            <status>running</status>
        </daemon>
        <daemon>
            <host>castelli</host>
            <command>master science database</command>
            <status>running</status>
        </daemon>
        <daemon>
            <host>kryten</host>
            <command>sah_validate4</command>
            <status>running</status>
        </daemon>
        ...
    </daemon_status>
        <database_file_states>
        <results_ready_to_send>563389</results_ready_to_send>
        <results_in_progress>1198237</results_in_progress>
        <workunits_waiting_for_validation>19</workunits_waiting_for_validation>
        <workunits_waiting_for_assimilation>16</workunits_waiting_for_assimilation>
        <workunits_waiting_for_deletion>0</workunits_waiting_for_deletion>
        <results_waiting_for_deletion>0</results_waiting_for_deletion>
        <transitioner_backlog_hours>-0.0002777</transitioner_backlog_hours>
    </database_file_states>
</server_status>
```

There are two ways to do this:

1. Copy the file server_status.php from html/ops to html/user. This works fine as long as you don't need any customization, and the DB queries in the page only take a few seconds (the page is cached, so they are done infrequently).
2. Write a periodic script that generates the 2 pages as files, and put a script in user/server_status.php that echoes one file or the other.

62. Profile screening and UOTD selection

Users can create profiles containing text and/or pictures. These profiles are displayed in various ways: as a picture gallery, sorted by country, and sorted alphabetically. The default project web site also shows a **User of the Day** (UOTD) on its front page.

Depending on your project's requirements and resources, there are two main approaches to screening profiles and selecting UOTD. Both involve a 'profile screening page' used by project staff.

- **Loose screening**: Only profiles with nonzero recent average are screened. The UOTD is selected

randomly from among approved profiles. Profile pictures are shown independently of whether they have been screened. This is the default.

- **Tight screening**: All profiles are screened. A profile's picture is shown only if the project has approved it. The UOTD is selected randomly from among approved profiles that have nonzero recent average credit. To use tight screening, include the line

  ```
  <profile_screening>1</profile_screening>
  ```

  in your config.xml file.

You can customize the database queries used to select profiles for screening and for UOTD candidacy. To do so, define either of the functions

```
uotd_candidates_query()
profile_screen_query()
```

in your html/project/project.inc file. Each function must return a SQL query.

63. Web page caching

Some pages on your project's web site are accessed often and require lots of database access to generate. To keep this from bogging down your server, BOINC caches these pages. This cache is in PROJECT/html/cache/*; a one-level hashed directory hierarchy is used to deal with large-directory performance problems.

Caching configuration

The file html/project/cache_parameters.inc contains a number of parameters related to caching:

TEAM_PAGE_TTL	Cache life of team pages; default 1 hour
USER_PAGE_TTL	Cache life of user pages; default 1 hour
USER_HOST_TTL	Cache life of user host list; default 1 hour
USER_PROFILE_TTL	Cache life of profiles; default 1 hour
TOP_PAGES_TTL	Cache life of user/team/host lists; default 12 hours
INDEX_PAGE_TTL	Cache life of main page; default 1 hour
MAX_CACHE_USAGE	Max cache size; default 100 MB
MIN_FREE_SPACE	Min free space on device; default 100 MB
CACHE_SIZE_CHECK_FREQ	Check cache size on every N user accesses to cached pages; default 1000

Caching and translation

BOINC uses several web-page caching systems, which support language translation in different ways.

- **Pre-generated**: Pages are updated from time to time, and do not support translation. The system used for building profiles is a pre-generated cache.
- **Fullpage cache**: This cache system simply takes the output of a page and saves it for the future. It uses the `start_cache()` and `end_cache()` functions in `cache.inc`. The pages may not be translation-aware (otherwise some users will see the wrong language).
- **Fullpage cache with translation**: You can make the language part of the cache filename. (To do this, you need to adapt the code in cache.inc). This can be inefficient because it stores a separate copy of the page for each language.
- **Object cache**: This stores the data used to create the page and recreates the page every time (using any language you'd like). Use `get_cached_data()` in `cache.inc`. This is perfect for pages that are accessed commonly and by people from many nationalities (currently the top-X pages support it).

If something shows up in the wrong language it's probably because a page that was previously not being translated got translation abilities but wasn't moved to the proper cache type. Also if a page-piece that is included is now translatable, all pages that make use of this piece should now either use fullpage caching with translation, or use an object cache (nicer but takes a few more lines of coding).

64. Recruiting and retaining volunteers

Contents

The following is a list of suggestions for getting more people to participate in your project. Much of this is based on the results of the BOINC user survey; study this yourself.

Most of these suggestions involve writing prose to be read by the general public. If (like many scientists) you are not good at this or avoid doing it, find someone who is good at it. This could be one of your students, a friend of a friend, or a professional writer.

English is the most widely-spoken language among BOINC participants, and you should probably use it as the main language for web materials and email. BOINC provides mechanisms for Web site translation; it's generally easy to get volunteers to do this.

Project web site

Your project's web site has a large role in attracting participants. Some suggestions:

- Present your project's credentials: the educational credentials of its leaders, its research track record, and the status of its institution.
- Describe what your project is doing: its high-level scientific goals, its methods, the details of the computation being done using volunteers, and the (non-distributed) computations that precede and follow this. How will your research affect the lives of everyday people now and/or 50 years from now?
- Who owns the intellectual property that arises from volunteer computations? Will it be released to the public? When, and under what terms?
- Show all the scientific results of the computation so far, and any publications that arise from these results. (Rosetta@home and Folding@home provide good examples of this). Announce new results and publications on the News column. Make sure your News column is being properly published as an RSS feed.
- Give some personal information about your team members: their names, background, interests, and preferably a photograph. This will 'humanize' your project in the eyes of potential participants.
- Take an active role in your web site's message boards. Read them frequently, and respond quickly to any negative threads that arise. Make a periodic posting giving 'insider info' on your project.
- Make sure your the web site has clear navigation, so that the above information is easy to find from the front page. Do a user study - show your web site to a strangers, ask them to browse it and/or to find particular information, and get their feedback (you may be surprised).
- If possible, create a graphical identity (logo, color scheme, etc.) for your project. Your web site should project professionalism and inspire confidence and interest in prospective volunteers.

Publicity

The world will not beat a path to your door. You need to work hard to spread the word about your project.

- Get in the mass media (newspapers, magazines, radio, television) as much as possible. If your institution has a PR director or media spokesperson, contact them while you're developing your project, and again any time your project has major news. If no such person is available, call local media outlets yourself.
- Exploit existing organizational relationships. If you work at a University, try to get your project running on the PCs in the teaching labs, and on the PCs of students, faculty and staff. If you have connections with organizations with PR capabilities (i.e., web sites or newsletters), enlist their support for your project, and get them to publicize it. A typical example: professional organization in your subject area.
- The BOINC web site will generally announce new projects. Also, make sure your project is listed on account managers like GridRepublic and BAM!
- Use the web. Announce your project in forums like Slashdot, and on the message boards of the major cross-project teams like BOINC Synergy, Overclockers UK, Team Anandtech, etc.

Email-based mechanisms

BOINC provides PHP-based tools for sending three types of email to participants:

- **Newsletters**. These are periodic (perhaps every few months) and are sent to all participants. Typically you would use them to summarize your project's results, to discuss its future plans, to make announcements, etc.
- **Reminders**. These are sent to participants who seem to have stopped computing to your project, or who signed up but never got any credit. Typically they would be short messages, encouraging the participant to take a specific action.
- **Friend-to-friend**. These are sent by participants to their friends and family, to tell them about your project and urge them to join. The sender can add an optional message.

Effective use of all types of email is critical to maintaining and growing your participant base. In the absence of any email, participation typically decreases by a few percent every month. **BOINC supplies the framework, but you must write the actual emails, or modify BOINC's samples as needed for your project.**
.

The newsletter and reminder scripts provide the following features:

- They let you send different emails to different 'classes' of participants. For example, you can send a different newsletter to participants who haven't computed for your project in a while.
- They let you personalize emails, e.g. by inserting the participant's name or their total credit.
- They provide a mechanism for inserting a secure 'opt-out' link. Note: You should ALWAYS include an 'opt-out' link at the bottom of emails (both HTML and text). In may be illegal for you to do a mass email without one. Make sure you test this link.

The scripts requires that you use PHPMailer, is a PHP function for sending mail that's more full-featured than the one built into PHP. Download it, put it in html/inc, and set the USE_PHPMAILER, PHPMAILER_HOST, and PHPMAILER_MAILER variables in your project.inc file.

All of the tools let you send multipart HTML/text messages. We recommend that you use this feature - and HTML message can include your logo and/or institutional insignia, can include hyperlinks, and can look more attractive.

The general procedure for using each scripts is:

- Create a directory (mass_email, reminder_email, or ffmail) in your html/ops/ directory. In that directory, create separate files for the text body template, HTML body template, and subject line to be sent to each class of participants. NOTE: the HTML files are optional; if you leave them out, text-only emails will be sent.
- Run the script in testing mode (see below) to ensure that the emails are as you intend.
- Once testing is complete, run the script in production mode. Typically, the newsletter script is run from the command line. The reminder script is typically run as a periodic task, every 24 hours or so.

The newsletter and reminder scripts use the recent-average credit (expavg_credit) field in the user table. To make sure this value is accurate, run update_stats manually if you're not running it as a periodic task.

Personalizing emails

The newsletter and reminder scripts replace the following macros in your email bodies (both HTML and text):

<name/>	User name
<create_time/>	When account was created (D M Y)
<total_credit/>	User's total credit
<opt_out_url/>	URL for opting out (this URL includes a salted version of the participant's account key, and so is different for every participant).
<lapsed_interval/>	The number of days since user's client contacted server (defined only for lapsed users, see below).
<user_id/>	The user ID (use this to form URLs)

Avoiding spam filtering

Your email is less likely to be rejected by spam filters if:

- Your HTML and text versions have the same text.
- Your HTML version either contains no images, or has at least 400 words.

Newsletters

The script **html/ops/mass_email_script.php** is for sending email newsletters. The script categorize participants as follows:

- **Failed**: zero total credit. These people failed to download and install the client software, or failed to get it working (e.g. because of proxy problems) or uninstalled it before finishing any work.
- **Lapsed**: nonzero total credit but recent average credit < 1: These people did work in the past, but none recently.
- **Current**: recent average credit >= 1. These are your active participants.

To use the script, create the following files in html/ops/mass_email:

failed_html HTML message sent to failed users. Example:

```
<html>
<body bgcolor=ffffcc>
Dear <name/>:
<p>
Test Project continues to do pioneering computational research
in the field of Submandibular Morphology.
In recent months we have discovered over
17 new varieties of Frombats.
<p>
Our records show that you created a Test Project account
on <create_time/> but that your computer
hasn't completed any work.
Possibly you encountered problems
installing or using the software.
Many of these problems have now been fixed,
and we encourage you to visit
<a href=http://a.b.c>our web site</a>,
download the latest version of the software, and try again.
<p>
<font size=-2>
To not receive future emails from Test Project,
<a href=<opt_out_url/>>click here</a>.
</font>
</td></tr></table>
</body>
</html>
```

failed_text Text message sent to failed users. Example:

```
Dear <name/>:

Test Project continues to do pioneering computational research
in the field of Submandibular Morphology.
In recent months we have discovered over
17 new varieties of Frombats.

Our records show that you created a Test Project account
on <create_time/> but that your computer
hasn't completed any work.
Quite possibly you encountered problems
installing or using the software.
Many of these problems have now been fixed,
and we encourage you to visit
<a href=http://a.b.c>our web site</a>,
download the latest version of the software, and try again.

To not receive future emails from Test Project, visit
<opt_out_url/>
```

email_failed_subject Subject line sent to failed users. Example: 'Test Project News'.

lapsed_html HTML message sent to lapsed users

lapsed_text Text message sent to lapsed users

lapsed_subject Subject line sent to lapsed users

current_html HTML message sent to current users

current_text Text message sent to current users

current_subject Subject line sent to current users

Testing

Test your email before sending it out to the world. As distributed, mass_email_script.php has the following variables defined near the top:

```
$testing = true;
```

Set it to false to actually send emails (rather than just print to stdout).

```
$userid = 1;
```

If this is nonzero, email will be sent to the given user ID; Otherwise it will be sent to all users.

To start, set $userid to the ID of your own user record. Run the script by typing

```
php mass_email_script.php
```

It will print (to stdout) the contents of all three email types (failed, lapsed, and current). Verify that the subject, HTML and text are correct.

Then set $testing = false and run the script again. You'll get three emails; check them.

Then set $testing = false and $userid = 0, create an empty file called **mass_email/log** (see below), and run the script. You'll get voluminous output to stdout, but no emails will be sent. Control-C it quickly if you want. Make sure that each user is being sent the right type of email.

When you're sure that everything is correct, set $testing = false, set **mass_email/log** to empty, and run the script. It will now send mass emails. Depending on the size of your user table, it may take hours or days to complete. You can control-C it and restart whenever you want; it automatically picks up where it left off (see below).

Checkpoint/restart

mass_email_script.php manages checkpoint/restart when dealing with large numbers of participants. Mails are sent in order of increasing user ID. The file **mass_email/log** has a list of IDs that have been processed. On startup, the script reads this file, finds the last entry, and starts from there.

If you are starting a mass email from the beginning, empty the file **mass_email/log**; i.e.

```
truncate mass_email/log
```

Reminder emails

The script **html/ops/remind.php** is for sending reminder emails. The script categorizes users as follows.

- **Failed**: the account was created at least 14 days ago, has zero total credit, and hasn't received a reminder email in 30 days. These people typically either had a technical glitch, or their hardware and/or preferences didn't allow sending them work, or the application crashed on their host. The reminder email should direct them to a web page that helps them fix these problems.
- **Lapsed**: the user has positive total credit, hasn't done a scheduler RPC in the past 60 days, and hasn't been sent a reminder email in the past 30 days. They probably stopped running BOINC or detached this project. The reminder email should gently prod them to start running BOINC and attach to this project again.

The numbers 14, 30, and 60 are all parameters in the script; edit it to change them.

To use the script, create the following files in **html/ops/reminder_email/**:

failed_html	HTML message sent to failed users
failed_text	Text message sent to failed users
failed_subject	Subject line sent to failed users
lapsed_html	HTML message sent to lapsed users
lapsed_text	Text message sent to lapsed users
lapsed_subject	Subject line sent to lapsed users

remind.php can be run as often as you like. We recommend running it every hours, specifying it as a task in config.xml. When it sends email to a user, it stores the time in their database record, and won't send them another email for at least 30 days. For this reason, it has no checkpoint/restart mechanism.

The procedure for testing your reminder email is similar to that for email newsletters (see above).

Friend-to-friend emails

The web page ffemail_form.php lets users send emails to their friends. To use this feature, you must create the following files in **html/ops/ffmail/**:

subject	The subject line used for friend-to-friend emails
html	HTML template for friend-to-friend emails
text	Text template for friend-to-friend emails

Samples are supplied for each of these. The following macros are substituted in the message bodies:

<fromname/>	The name of the sender
<toname/>	The name of the recipient

<comment/>	The comment supplied by the sender

65. GUI URLs

GUI URLs is a mechanism that projects to pass URLs to the client, for display as hyperlinks in the GUI. These links will be shown when the project is selected in the **Projects** tab.

To use this feature, include a file 'gui_urls.xml' in the project root directory, with the following form:

```
<gui_urls>
    <gui_url>
        <name>Your account</name>
        <description>View your account information and credit totals</description>
        <url>http://foo.project.com/show_user.php?userid=<userid/></url>
    </gui_url>
    <gui_url>
        <name>Help</name>
        <description>Get help about SETI@home</description>
        <url>http://foo.project.com/help.php</url>
    </gui_url>
    <ifteam>
        <gui_url>
            <name>Team</name>
            <description>Info about <team_name/></description>
            <url>http://foo.project.com/team_display.php?teamid=<teamid/></url>
        </gui_url>
    </ifteam>
    ...
</gui_urls>
```

Each entry describes a GUI URL. These URLs (macro-substituted as described below) will be sent to client hosts in the reply to scheduler RPCs. Team-specific entries should be enclosed in `<ifteam>`; they will be sent only if the user belongs to a team.

The components of a `<gui_url>` element are:

name	A short name, used e.g. as a button name or menu item
description	An explanation, used e.g. as a rollover popup
url	The URL

All items are macro-substituted as follows:

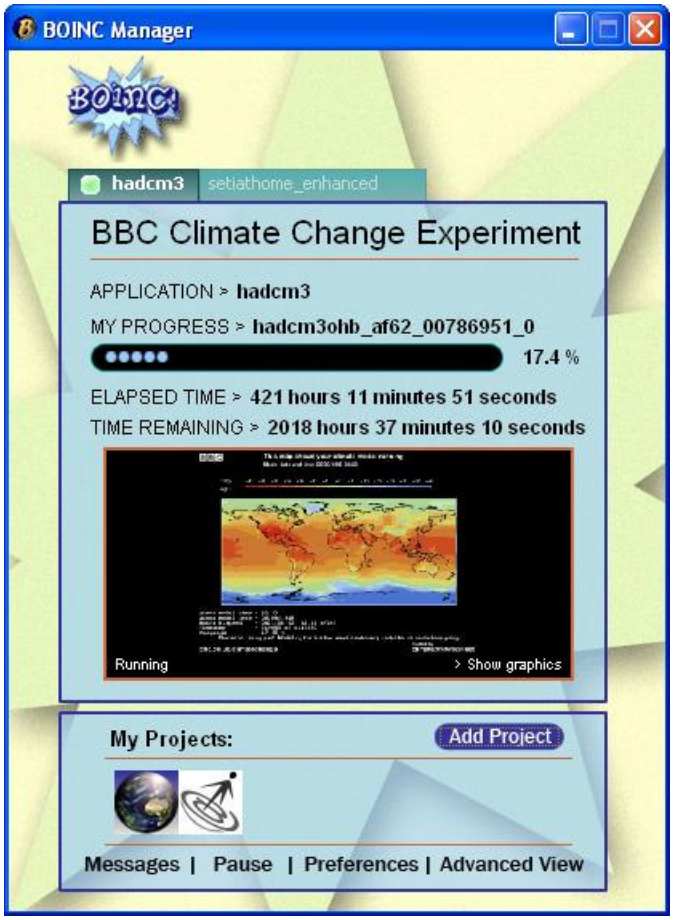
<userid/>	The user ID
<user_name/>	The user name
<teamid/>	The team ID
<team_name/>	The team name
<hostid/>	The host ID
<authenticator/>	The user's account key

66. Project graphics in the BOINC simple GUI

The 'simple GUI' available in versions 5.8+ of the BOINC Manager uses graphical representations of projects and applications:

- The project is represented by a 40x40 pixel icon.
- Each application is represented by a sequence of images, each up to 290x126 pixels. These are shown as a slideshow, changing once every few seconds.

For example, in the following screenshot of the simple GUI, the two icons at the bottom represent CPDN and SETI@home, and the earth-map image in the middle represents the particular CPDN application that is currently running.



Specifying project files

Project graphics files are specified in a configuration file **project_files.xml** that you put in your project's root directory. This file specifies a set of 'project files' that will be automatically downloaded to clients (this can be used for purposes other than graphics).

The format of 'project_files.xml' is:

```
<file_info>
    <name>X</name>
    <url>X</url>
    <md5_cksum>X</md5_cksum>
</file_info>
...
<project_files>
    <file_ref>
        <file_name>X</file_name>
        <open_name>X</open_name>
    </file_ref>
    ...
</project_files>
```

For each file, this specifies:

- its URL (where to download it from)

- its physical name; it will be stored in the project directory on the client under this name.
- its MD5 checksum (use 'openssl dgst' to get this).
- its logical name; a 'soft link' file will be created with this name, linking to the physical name.

All file_info records must appear before the project_files record.

As with all BOINC files, project files are immutable. If you want to change the contents of a file, you must use a new physical name.

Project Icon

A project icon is a 40x40 image, PNG, JPG, GIF, or BMP format.

An example would look like this:

```
<file_info>
    <name>stat_icon_01.png</name>
    <url>http://www.example.com/download/stat_icon_01.png</url>
    <md5_cksum>186c5385c8f2a48ccc7e4f64251fcda1</md5_cksum>
</file_info>
<project_files>
    <file_ref>
      <file_name>stat_icon_01.png</file_name>
      <open_name>stat_icon</open_name>
    </file_ref>
</project_files>
```

Two things to note here:

- The '01' in stat_icon_01.png is used for versioning.
- The physical name for the project icon is 'stat_icon_01.png' while the logical name for the project icon is 'stat_icon'. The manager looks for 'stat_icon' and resolves it to a physical name.

Application Slideshow

You can have one or more images displayed in the Simple GUI when BOINC is running one of your apps. Each image can have a height up to 126px and a width up to 290px, and can be any of the following image types: PNG, JPG, GIF, and BMP.

An example would look like this:

```
<file_info>
    <name>slideshow_exampleapp_01_01.png</name>
    <url>http://www.example.com/download/slideshow_exampleapp_01_01.png</url>
    <md5_cksum>186c5385c8f2a48ccc7e4f64251fcda1</md5_cksum>
</file_info>
<file_info>
    <name>slideshow_exampleapp_02_01.png</name>
    <url>http://www.example.com/download/slideshow_exampleapp_02_01.png</url>
    <md5_cksum>3b262da3d69d6b9eb55add88b66cdab4</md5_cksum>
</file_info>
</file_info>
<project_files>
    <file_ref>
      <file_name>slideshow_exampleapp_01_01.png</file_name>
      <open_name>slideshow_exampleapp_01</open_name>
    </file_ref>
    <file_ref>
      <file_name>slideshow_exampleapp_02_01.png</file_name>
      <open_name>slideshow_exampleapp_02</open_name>
    </file_ref>
</project_files>
```

In this example:

```
slideshow_exampleapp_02_01.png
```

'slideshow_' labels it as a slideshow file, 'exampleapp' is the application short name, '02' is the index of the slide within the slideshow, and 01 is the version of the file.

Example

Here is the 'project_files.xml' file SETI@home is using:

```
<file_info>
    <name>arecibo_181.png</name>
    <url>http://setiathome.berkeley.edu/sg_images/arecibo_181.png</url>
    <md5_cksum>f9b65230a594098d183d2266511bc648</md5_cksum>
</file_info>
<file_info>
    <name>sah_40.png</name>
    <url>http://setiathome.berkeley.edu/sg_images/sah_40.png</url>
    <md5_cksum>5791ba1be2d33eaa5f90ecf5de89a53d</md5_cksum>
</file_info>
<file_info>
    <name>sah_banner_290.png</name>
    <url>http://setiathome.berkeley.edu/sg_images/sah_banner_290.png</url>
    <md5_cksum>39839286db7f580bef5377322d15ed35</md5_cksum>
</file_info>
<file_info>
    <name>sah_ss_290.png</name>
    <url>http://setiathome.berkeley.edu/sg_images/sah_ss_290.png</url>
    <md5_cksum>caf95504208aedd6ac6d82201e2fd8b1</md5_cksum>
</file_info>
<project_files>
    <file_ref>
      <file_name>sah_40.png</file_name>
      <open_name>stat_icon</open_name>
    </file_ref>
    <file_ref>
      <file_name>sah_ss_290.png</file_name>
      <open_name>slideshow_setiathome_enhanced_00</open_name>
    </file_ref>
    <file_ref>
      <file_name>arecibo_181.png</file_name>
      <open_name>slideshow_setiathome_enhanced_01</open_name>
    </file_ref>
    <file_ref>
      <file_name>sah_banner_290.png</file_name>
      <open_name>slideshow_setiathome_enhanced_02</open_name>
    </file_ref>
</project_files>
```

67. Exporting credit statistics with db_dump

The program db_dump generates XML files containing your project's credit data. It should be run every 24 hours. Include an entry like

```
    <tasks>
        <task>
            <cmd>db_dump -d 2 -dump_spec ../db_dump_spec.xml</cmd>
            <output>db_dump.out</output>
            <period>24 hours</period>
        </task>
    </tasks>
```

in your config.xml file. Make sure the file db_dump_spec.xml is in your project's root directory.

The XML files are written to html/stats/, and the old stats/ directory is renamed to stats_DATE. This clutters up your html/ directory; if you don't like this, create a directory html/stats_archive/ and add the line

```
<archive_dir>../html/stats_archive</archive_dir>
```

to your db_dump_spec.xml file.

68. Integrating BOINC projects with Grids

Researchers are CERN have set up a system where submitted jobs are sent either to a BOINC project or to a GRAM job manager. They developed two utilities, kill_wu and poll_wu, to support this. They are in the boinc/tools directory. Contact Christian Søttrup (chrulle at fatbat.dk) for more info.

The Lattice project from the University of Maryland is developing a Grid system that integrates Globus,

BOINC, and several other software components.

69. Versions of BOINC

The BOINC software (including client and server components) evolve over time. There are a number of pairwise interactions in which version mismatches could cause problems:

- RPC from core client to scheduling server.
- RPC from core client to file upload handler.
- Interface between core client and application.
- Interface between BOINC DB and all BOINC back-end components.
- The parsing of the core state file by the core client.

Each BOINC software component has a version consisting of three integers: major, minor, and release.

Some changes to the BOINC server software may involve changes to the BOINC database (e.g. adding a new table or field). Such releases will include SQL script for modifying an existing database in-place.

70. Configuring MySQL for BOINC

A fast-and-easy script that makes recommendations for tuning server variables is here.

Introduction

The note discusses how MySQL may be configured for BOINC Projects. BOINC-based projects have varying DB traffic characteristics and this note relates to our experiences with SETI@home, so it may not be entirely applicable to all projects. SETI@home currently uses MySQL 4.0+ and we expect to upgrade to 4.1 shortly and 5.0 later. Our project uses only a single instantiation of the MySQL code file and this note does not discuss the operation of multiple instances of MySQL on a single server.

All MySQL products and documentation are available at http://www.mysql.com/. Our experience has been of using MySQL with Sun Solaris and Linux OSes. MySQL on MS Windows or Mac OS X may be somewhat different.

MySQL DB Engines (or Table Types)

General

The MySQL software comprises a number of DB engines. For SETI@home DB only 2 are used, Innodb and MyISAM. They have different features and are used according to the performance requirements of the project. One can use all of the different engines (or table types) or just a single one in a MySQL DB, just depending on the query activity against each table in the project among other.

MySQL software is available in 32 bit and 64 bit binaries for downloading. Using 32 bit MySQL requires that all RAM resources that are assigned to the various DB engines, must sum to no more than 2GB of RAM. There is no such limitation with 64 bit MySQL and large amounts of RAM help Innodb performance.

MyISAM

The MyISAM engine requires the least amount of computer resources can be used where there is a low DB activity requirement. For example with query rates lower that 5/sec this table type may be adequate. Also if one does not have a dedicated DB server this may be a good choice for all the tables since it consumes much less computer resources. It has the advantages of allowing long text indices against tables which Innodb does not allow.

MyISAM creates an OS file for each table and one for all the indices related to the specific table (and another

for the table format info).

On the other hand it tends to suffer from consistency glitches so will occasionally trash indices and will need rebuilding. In commercial banking environments it would not be a good idea to keep account balances in this table type since there is no guarantee that transactions even if completed and printed will remain in the DB. MyISAM updates its tables synchronously and uses memory locks to avoid data collisions. In SETI@home, MyISAM is used for the forum tables and logging that have relatively low query rates.

Innodb

The Innodb engine is used for most of the tables in SETI@home project. It processes multiple simultaneous queries against its tables. It is a versioning DB engine that holds an image of the table at the start of a query and maintains it until that query is completed. Other updates are allowed during queries and in general for short queries there is no problem. Innodb uses the Innodb log to store changes to its tables until it flushes these changes to the actual tables at syncpoints. If for any reason there is a server event that causes a system failure, Innodb will use this log to recover the Innodb tables to consistency. There are a minimum of 2 transaction log files with a total maximum size of 4GB.

Innodb tables/indices are usually stored in large OS physical files and the tables and indices are managed internally within these OS/Innodb files. It is important that these files are located on high performance devices. The transaction log files should be located on independent high performance media (away from the Innodb files) for sustained high transaction rates. At DB shutdown all modified buffers have to be flushed into the transaction logs before MySQL goes away, so slow performance drives for the transaction log could delay shutdown for over 30 minutes when there are a large number of .modified buffers. to be flushed.

Physical Requirements

CPU

Assuming the need for more than 70,000 users and 250K hosts with an average workunit turnaround of about 10 hours then one should get an Opteron dual-core class CPU. It is a 64-bit architecture and can access up to 32GB of RAM. It is qualified to run Solaris, Linux and Windows XP 64-bit (?) . There are 64-bit versions of MySQL for Linux and Solaris OSes.

This is by no means the only hardware that will work with BOINC/MySQL, however SETI@home uses this type of hardware and serves over 350K user and over 630K hosts. If your requirements are smaller, then many 32bit hardware and OSes may be perfectly adequate.

RAM

The RAM requirement is related to the number of active subscribers who are expected to volunteer for the project and the number of threads that will be connected to the MySQL server. We recommend a minimum of 2GB dedicated to MySQL for about 20,000 . 30,000 volunteers growing to servers with much larger RAM sizes, say 6GB for up to 450K volunteers. This is also related to disk IO rates that are available for use by the data and log files. For example Innodb will store modified data in RAM until a syncpoint at which time data is flushed to disk; during this time update transactions are paused until the flush is completed. If there is large RAM and slow disk IO, the pause can last for several minutes. A similar delay can be noted when attempting to shutdown the project database when all the modified buffers must be flushed to disk before MySQL will shutdown, this delay could be 30 minutes or more.

IO Subsystem

Assuming a high performance requirement of more than 200 DB queries/sec there should be separate controllers for for the data and the log files. In the case of Innodb log files it is very important that they are on very reliable media for example mirrored (RAID 1) drives. The tables and indices require wide band or high throughput disk configuration such as RAID 10.

Some consideration should be given to having online spare disk drives since this will help to minimize down times in case of failures.

Normal Operations

General

For normal operations or production there are some considerations that should be addressed to enable the project personnel to provide reliable service. For example there should be a reliable power supply with UPS protection to avoid uncontrolled shutdowns. The temperature of the hardware operations room should be regulated to hardware specifications to avoid premature aging/failure of hardware components.

And the MySQL software has to be set up to take advantage of the hardware resources that are available.

Config File (my.cnf)

The config file needs to be set up for production environment. MySQL has defaults for where it allocates the files that it needs; where they are placed depends on the OS on which it is running. For greater control, space management and performance the user should define where these files are assigned. For example the base data directory for MySQL tables etc in Linux is /var/lib/MySQL. For SETI@home we assigned this to directory to another data partition /mydisks/a/apps/mysql/data/, to ensure that there was enough space and performance. It made it easy to do physical backups without including additional files that were not related to the database.

Here are some other file directory assignments for the SETI@home environment:

```
innodb_data_home_dir = /mydisks/a/apps/mysql/data/
innodb_data_file_path = ibdata1:16G;ibdata2:16G;ibdata3:16G;
    ibdata4:16G;ibdata5:16G; ibdata6:16G;ibdata7:16G;ibdata8:16G;
    ibdata9:16G;ibdata10:16G;ibdata11:16G;ibdata12:16G;
innodb_log_group_home_dir = /mydisks/a/apps/mysql/mysql_logs/innodb_logs/
innodb_log_arch_dir = /mydisks/a/apps/mysql/mysql_logs/innodb_logs/
```

Example of a MySQL config file:

```
[mysqld]
#datadir=/var/lib/mysql
#datadir=/home/mysql/data/
datadir=/mydisks/a/apps/mysql/data/
#log-bin      ##/// this comment line disables replication
log-slow-queries = /mydisks/a/apps/mysql/jocelyn_slow.log
server-id       =       13
socket=/tmp/mysql.sock
skip-locking
set-variable    = delay_key_write=all
set-variable    = key_buffer= 750M
set-variable    = max_allowed_packet=2M
set-variable    = table_cache=256
set-variable    = sort_buffer=2M
set-variable    = record_buffer=2M
set-variable    = myisam_sort_buffer_size=512M
set-variable    = query_cache_limit=2M
set-variable    = query_cache_size=16M
set-variable    = thread_cache=128

# Try number of CPU's*2 for thread_concurrency
set-variable    = thread_concurrency=8
set-variable    = max_connections=256
set-variable    = max_connect_errors=1000

## more changes for slave replicant
#master-host     = xxx.ssl.berkeley.edu
#master-user     = slavexxx11
#master-password = masterpwxxx11
#replicate-do-db        =       SETI_BOINC
#replicate-ignore-db    =       mysql

# Uncomment the following if you are using Innobase tables
innodb_data_home_dir = /mydisks/a/apps/mysql/data/
innodb_data_file_path = ibdata1:16G;ibdata2:16G;ibdata3:16G;
    ibdata4:16G;ibdata5:16G; ibdata6:16G;ibdata7:16G;ibdata8:16G;
    ibdata9:16G;ibdata10:16G;ibdata11:16G;ibdata12:16G;
innodb_log_group_home_dir = /mydisks/a/apps/mysql/mysql_logs/innodb_logs/
innodb_log_arch_dir = /mydisks/a/apps/mysql/mysql_logs/innodb_logs/
set-variable = innodb_mirrored_log_groups=1
set-variable = innodb_log_files_in_group=4
```

```
set-variable = innodb_log_file_size=1000M
set-variable = innodb_log_buffer_size=16M
set-variable = innodb_flush_method=O_DIRECT
set_variable = innodb_fast_shutdown=1
innodb_flush_log_at_trx_commit=0
innodb_log_archive=0
set-variable = innodb_buffer_pool_size=4584M
set-variable = innodb_additional_mem_pool_size=8M
set-variable = innodb_file_io_threads=64
set-variable = innodb_lock_wait_timeout=50

[mysql.server]
user=mysql
basedir=/mydisks/a/apps/mysql

[safe_mysqld]
err-log=/mydisks/a/apps/mysql/jocelyn.err
pid-file=/mydisks/a/apps/mysql/jocelyn.pid
```

Monitoring

MYTOP

During normal operations it is useful to monitor the MySQL IO traffic, memory usage and connection activity to various client applications. Mytop application script give useful realtime status for the MySQL engine. Here is a sample of the first lines of its output:

```
MySQL on localhost (4.0.23-max-log)
 up 18+00:32:55 [10:50:21]
 Queries: 641.7M  qps:  432 Slow:   71.4k         Se/In/Up/De(%):    51/01/43/03
     qps now:  382 Slow qps: 0.0  Threads:  413 (   2/  28) 43/01/46/09
 Cache Hits: 58.2M Hits/s: 39.2 Hits now:  17.3  Ratio: 17.9% Ratio now: 10.6%
 Key Efficiency: 99.4%  Bps in/out:  1.7k/ 1.6k   Now in/out: 63.5k/338.1k
```

It shows the historic queries/sec is 432 qps and the current sample was measured at 382 qps. The query cache hit rate is 17.9% historically and for the current sample period it is 10.6% and the cache fulfillment rate is 39.2 qps.

Useful Innodb information from Mytop is shown towards the end of the display for Innodb. The buffer pools information in given in number of pages that are 16KB in size. See example below:

```
--------
FILE I/O
--------
I/O thread 0 state: waiting for i/o request (insert buffer thread)
I/O thread 1 state: waiting for i/o request (log thread)
I/O thread 2 state: waiting for i/o request (read thread)
I/O thread 3 state: waiting for i/o request (write thread)
Pending normal aio reads: 0, aio writes: 0,
 ibuf aio reads: 0, log i/o's: 0, sync i/o's: 0
Pending flushes (fsync) log: 0; buffer pool: 0
1470930 OS file reads, 543461 OS file writes, 53800 OS fsyncs
1 pending preads, 0 pending pwrites
228.88 reads/s, 21594 avg bytes/read, 185.98 writes/s, 13.50 fsyncs/s
-------------------------------------
INSERT BUFFER AND ADAPTIVE HASH INDEX
-------------------------------------
Ibuf for space 0: size 335, free list len 283, seg size 619,
219535 inserts, 211776 merged recs, 45660 merges
Hash table size 9097667, used cells 2711301, node heap has 4751 buffer(s)
1573.54 hash searches/s, 5752.12 non-hash searches/s
---
LOG
---
Log sequence number 557 540674217
Log flushed up to   557 540451369
Last checkpoint at  556 4020363027
0 pending log writes, 0 pending chkp writes
39114 log i/o's done, 0.70 log i/o's/second
----------------------
BUFFER POOL AND MEMORY
----------------------
Total memory allocated 5032392104; in additional pool allocated 8386560
Buffer pool size   280576
Free buffers       0
Database pages     275825
Modified db pages  186393
```

```
Pending reads 1
Pending writes: LRU 129, flush list 0, single page 0
Pages read 2143598, created 23058, written 694488
301.17 reads/s, 4.40 creates/s, 216.68 writes/s
Buffer pool hit rate 991 / 1000
--------------
ROW OPERATIONS
--------------
6 queries inside InnoDB, 0 queries in queue
Main thread process no. 12155, id 1147140464, state: sleeping
Number of rows inserted 9780, updated 1039701, deleted 60084, read 159846476
0.10 inserts/s, 374.56 updates/s, 63.69 deletes/s, 1116.99 reads/s
----------------------------
END OF INNODB MONITOR OUTPUT
============================
```

IOSTAT:

Iostat is the UNIX type utility that provides a display of the IO statistics for peripherals on a server or workstation. For continuous displays of extended information for all devices. Iostat should be invoke as follows:

```
Iostat .x .k 5
```

(this will produce an updated display every 5 seconds for all devices and give data in KB)

MySQLAdmin

This program is making changes and getting the status of various MySQL parameters. It is not interactive but can be made to repeat a given function by using number repeat option. For example

```
mysqladmin  extended-status  10
```

This will show the status display and repeat the display every 10 seconds. Adding the .r option will give followup displays that show delta differences with the first display values. Performance Tweaking

General

An often overlooked area of performance is the requirement for reliable power and air conditioning. Power failures can eliminate all the benefits accrued by careful planning for hardware and software installations. Experience is that unreliable power can lead to days of recovery with data loss and subscriber discontent. Similarly, insufficient cooling accelerates the aging of hardware components and can cause data corruption and downtime more frequently than the one would expect given the hardware specs.

```
There are several parameters in my.cnf that can be adjusted (within limits)
for better throughput.
Then the distribution of MySQL files to specified disk subsystems,
allocation of RAM and   Config: my.cnf options for files, RAM, IO options
```

MySQL Configuration

Multi threads, query caching

Files. Distribution

Innodb files, transaction log files, bin-log files, MyISAM data/index files

Slow Query Log

Turn on Slow Query log to monitor slow queries.

RAM Allocation

Innodb vs MyISAM

71. Controlling account creation

Under normal circumstances BOINC projects are open for participation by anybody who wants to contribute their computer to the project. There may be times, however, when a project needs to limit the creation of new accounts. BOINC offers two alternatives.

Disabling account creation

To disable all account creation, edit the project configuration file config.xml and add to it the element:

```
<disable_account_creation>1</disable_account_creation>
```

This disables account creation via any mechanism (the client, the web, or account managers). You can momentarily remove this element while you create accounts.

Restricting account creation via 'invitation codes'

It is also possible to restrict account creation to only those who present a secret 'invitation code'. In this case an account can only be created via the web pages, not via the client or an account manager.

To use this mechanism you need to add to the file html/project/project.inc a definition for a PHP pre-processor symbol INVITE_CODES containing the allowed invitation codes. A simple example is:

```
define('INVITE_CODES', '/xyzzy/');
```

This allows account creation only if the user enters the invitation code 'xyzzy' (without any quotes). The pattern in INVITE_CODES is compared to the user's input as a Perl-Compatible Regular Expression (PCRE), so don't forget the enclosing slashes. A more complicated example is:

```
define('INVITE_CODES', '/yohoho|blunderbuss|!grog4U/');
```

In a PCRE vertical bars separate alternatives, so this pattern just allows someone to create an account if they enter any of the words 'yohoho', 'blunderbuss', or '!grog4U'. More complex pattern matching is possible, though not required.

The security of this mechanism depends on how you distribute the invitation codes. If you write the code on the whiteboard in your lab then only someone with access to that room can use it. If you send it out to a mailing list then only members of that list can use it (until someone shares it with someone else who is not on the list). The goal here is not strict security so much as a way for a new project to limit account creation to a restricted set of users while the project is getting started.