Diplomarbeit

Distributed Computing for Problems from Combinatorial Geometry

Bernhard Kornberger

Institut für Softwaretechnologie
Technische Universität Graz
Vorstand: Univ.-Prof. Dipl.-Ing. Dr. techn. Franz Wotawa

Begutachter und Betreuer: DI. Dr. Oswin Aichholzer
Graz, im Mai 2005

Kurzfassung

Viele wissenschaftliche Forschungsprojekte erfordern bei weitem mehr Rechenleistung als heutige Personal Computer bieten. Der Rechenzeitbedarf beträgt dabei oft mehrere hundert Jahre auf einem Einzelplatzrechner, und die Untersuchungen stoßen daher an die Grenzen des Machbaren.

Auf der anderen Seite stehen in den Instituten und Subzentren der TU-Graz hunderte PC's, deren Rechenleistung - etwa während der Nachtstunden, Textverarbeitung oder Internet-Recherchen - nicht einmal annähernd ausgenützt wird. Im Zuge dieser Diplomarbeit ist ein System für verteiltes Rechnen entstanden, das genau diese Rechenleistung nutzbar macht und der rechenintensiven, wissenschaftlichen Forschung zur Verfügung stellt.

Abstract

Research projects in an academic setting frequently require computing capacities by far exceeding those offered by modern personal computers. Some tasks require computing times of a hundred years or more, researchers consequently finding themselves confronted with the limits of feasibility. On the other hand, departments and computer rooms at the University of Technology Graz possess hundreds of computers. Their capacities are far from being fully exploited, especially during the night or while they are used for text processing or internet research work. During this diploma thesis a system for distributed computing evolved dedicated to bundling and making this idle computing power available for extensive scientific investigations.

Danksagung

Diese Diplomarbeit wurde im Studienjahr 2004/05 am Institut für Softwaretechnologie an der Technischen Universität Graz durchgeführt.

Ich danke Herrn DI. Dr. Oswin Aichholzer für die engagierte und kompetente Betreuung dieser Arbeit. Er hatte auch die Idee, parallel dazu zwei Projektarbeiten zu starten, in deren Rahmen meine Kollegen Bernd Haug und Thomas Uttenthaler Arbeit und Know-How in diese Diplomarbeit eingebracht haben. Gemeinsam konnten wir in zahlreichen Besprechungen Lösungen für verschiedene Probleme erarbeiten und das hat deutlich zur Qualität dieser Arbeit beigetragen. Die Fakultät für Informatik hat mir ein Förderstipendium gewährt, um für diese Arbeit einen Server anzuschaffen. Dieser steht nun als zentrales Element des entstandenen Systems am IST und hat es in diesem Umfang erst ermöglicht.

Dank gebührt auch meiner Familie für den Rückhalt, den sie mir bietet. Im besonderen meiner Schwester Angela, die mich motiviert hat, nach der Abendschule ein Studium zu beginnen. Meine Freundin (und seit kurzem auch Verlobte) Birgit Holzer hat während der sechs Jahre unserer gemeinsamen Studienzeit fest zu mir gehalten, und auch ihre Eltern und Großeltern waren immer für uns da. Meine Freunde möchte ich wissen lassen, daß ich es schätze, daß sie Verständnis hatten, daß ich zuletzt wenig Zeit hatte und ihnen danken für die wertvollen Gespräche, die Sichtweisen und Länder, die ich mit ihnen kennenlernen durfte.

Graz, im Mai 2005 Bernhard Kornberger

Contents

Chapter 1

Introduction

1.1 Motivation and objectives

Today, many computing tasks in academic settings require such enormous amounts of computing capacity that computing times in the order of hundreds of years would be required even where modern high-end PCs are used. As access to the small number of super computers available in the world today is limited, such tasks must currently be considered practically unfeasible. The objective of the present diploma thesis is to create a system distributing such tasks over several hundreds of clients connected within a network so that they can be executed within a few months using the bundled computing capacity of many computers.

For the purpose of this project, it is assumed that the tasks to be processed can be distributed in the form of small sub-tasks completely independent from each other. These sub-tasks (work units) will be distributed from a central server via a network to the connected computers (clients) which will execute the tasks during their respective idle periods and finally transfer the results to the server. The clients will consist of computers owned by the Department, computers used at the computer rooms of the TUG and external computers of persons wishing to support the research activities of the TUG. This project is implemented at the Department of Software Technology (IST) [17] under the name dIST ("distributed computing at IST") [11].

1.2 Structure

As the project dIST allows for a considerable number of different strategies as far as its practical implementation is concerned, I have, as a first step, evaluated existing solutions as outlined in Chapter 2. This evaluation has shown that implementation of a completely new solution will not be reasonable because of the extensive range of functionalities required. Instead, I have decided to use and modify an existing project. For this purpose, I have selected the middleware BOINC (Berkeley Open Infrastructure for Network Computing) which serves as a basis for different projects such as the well-known distributed computing project SETI [27]. A detailed description of BOINC can be found in Chapter 3 and Chapter 4. Chapter 5 contains a description of the functionalities added to BOINC so as to meet the requirements of our dIST project and indicates which sections of the BOINC Software have been changed and how the changes can, if required, be integrated into the source code. Chapter 6 contains step-by-step instructions for installation and operation of a BOINC server and the set-up of BOINC projects as well as detailed descriptions explaining how to create and administrate work units. Chapter 7, finally, contains a decription of real applications running on dIST.

Chapter 2

Evaluation

As a first step in the framework of this diploma thesis, I have evaluated existing software solutions for similar tasks so as to gain an overview of the mechanisms required to be incorporated in this type of software. This evaluation enabled me to identify the components required to be included in the software solution to be offered:

- The system to be proposed must consist of software for the server distributing the tasks as well as client software for at least Linux and Windows.
- The software must comprise a network functionality. In this respect, any necessity on the part of the server to connect to the clients must be prevented as many of the clients will have dynamic IPs and be inaccessible behind firewalls and NAT routers. Therefore, it is essential that all communication with the server is initiated by the clients themselves.
- The clients must be protected against attempts by third parties to illegally transfer executable code onto the systems of the clients. Additionally the data transferred must be protected against error and manipulation, thus rendering application of cryptography indispensable.
- Further effort will be required as a consequence of the fact that reliable availability of the clients cannot be assumed, which means that the clients may be switched off and/or disconnect from the network at any time without

prior announcement. Therefore, precautions must be taken in the form of a workunit administration feature reallocating overdue workunits.

- A wide variety of client configuration options must be provided so as to ensure optimal adaptation of the client and its respective characteristics to the system. It will, for example, be essential to determine the speed of the respective computer and to define, on the basis of the speed determined, the work volume to be allocated to it at a time.
- Additionally, a specific feature must be included to ensure that the server will allocate large workunits only to computers known to be active 24 hours a day such as, for example, the computers of the IST.

These considerations have led to the conclusion that implementing a totally new software meeting the requirements outlined above is not recommendable for two reasons: Firstly, a software project providing the range of functionalities specified above would be enormously extensive. Secondly, any newly developed software would need to pass a phase of troubleshooting and elimination of safety gaps, which would exceed the cooperation that can be expected from the clients. Therefore, I looked for an existing software solution offering as many of the features required for dIST as possible and additionally being licensed as Open Source Software (OSS) so that modifications are possible and permitted. The evaluation of different existing software solutions included below is not intended to be exhaustive but merely provides an overview of the software solutions and approaches currently available. It is based on the web presentations and papers on the projects indicated available as of summer 2004.

2.1 OpenMosix

OpenMosix [22] is (as opposed to the Mosix project) released under the GNU General Public License (GPL) [13]. OpenMosix is a kernel extension for single-system image clustering. Once OpenMosix is installed, the nodes in the cluster start talking to one another and the cluster adapts itself to the workload. Processes originating from any one node, if that node is too busy compared to others, can migrate to any other node. OpenMosix continuously attempts to optimize the resource allocation. Since all OpenMosix extensions are inside the kernel, every Linux application automatically and transparently benefits from the distributed computing concept of OpenMosix and there is no need to program applications specifically for OpenMosix. The cluster behaves much as does a Symmetric Multi-Processor (SMP), but this solution scales to well over a thousand nodes which can themselves be SMPs.

The project OpenMosix is still active, but its further development is proceeding very slowly as only the Linux kernel 2.4.22, as opposed to the up-to-date kernel versions 2.4.30 and 2.6.10, is supported. Additionally, the characteristics of OpenMosix hardly meet the dIST specifications so that OpenMosix does not represent an option for the project dIST. Nevertheless, it is a highly interesting project that may represent a useful resource for application on the level of the Department.

2.2 InteGrade

InteGrade [15] is a grid middleware architecture allowing execution of parallel applications in a distributed environment. It is based on state-of-the-art middleware technology such as the CORBA [10] industry standard for distributed object systems which facilitates interaction between system and applications via well defined IDL interfaces accessible from a large variety of programming languages and operating systems. InteGrade builds on shared commodity hardware, which is suitable for our project. In such a dynamic environment, scheduling the execution of applications is rather difficult as an idle resource may turn busy again without further notice. To minimize this problem, the InteGrade architecture includes a feature collecting and analyzing usage patterns, i.e. a mechanism that is able, on the basis of usage information and statistics, to determine the probability of an idle node turning busy again. This mechanism can help schedulers to predict whether an idle machine will stay idle for a significant amount of time or will be busy again in a few seconds.

In the following paragraphs I provide a summary of the description of the InteGrade architecture (see [16] for more details): InteGrade Grids are structured

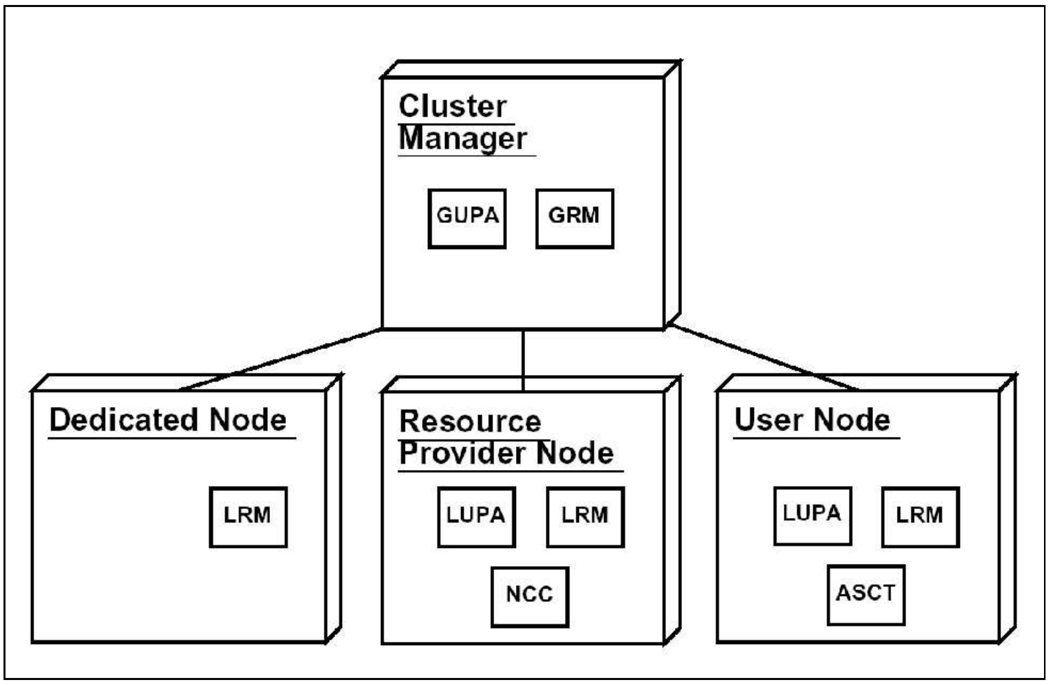

Figure 2.1: InteGrade Intra Cluster Architecture

in the form of clusters each consisting of groups comprising between one and approximately one hundred computers that may be shared workstations or machines dedicated to the grid. The clusters themselves are arranged in a hierarchy. There is a Cluster Manager representing one or more nodes responsible for managing the respective cluster and communicating with managers of other clusters. A User Node is a node belonging to a grid user who submits applications to the grid. A Resource Provider Node, typically a workstation, is a node that exports part of its resources, making them available to grid users. A Dedicated Node is a node reserved for grid computing. However, these categories may overlap: A node can be a User Node and a Resource Provider Node at the same time. The Local Resouce Manager (LRM) and the Global Resource Manager (GRM) cooperatively handle intra-cluster resource management. The LRM is executed in each cluster node, collecting information about the node status such as memory, CPU, disk, and network utilization. The LRMs periodically send the information thus obtained to the GRM which uses it for scheduling within the cluster (Information Update Protocol). The GRM and LRMs also collaborate in the Resource Reser-

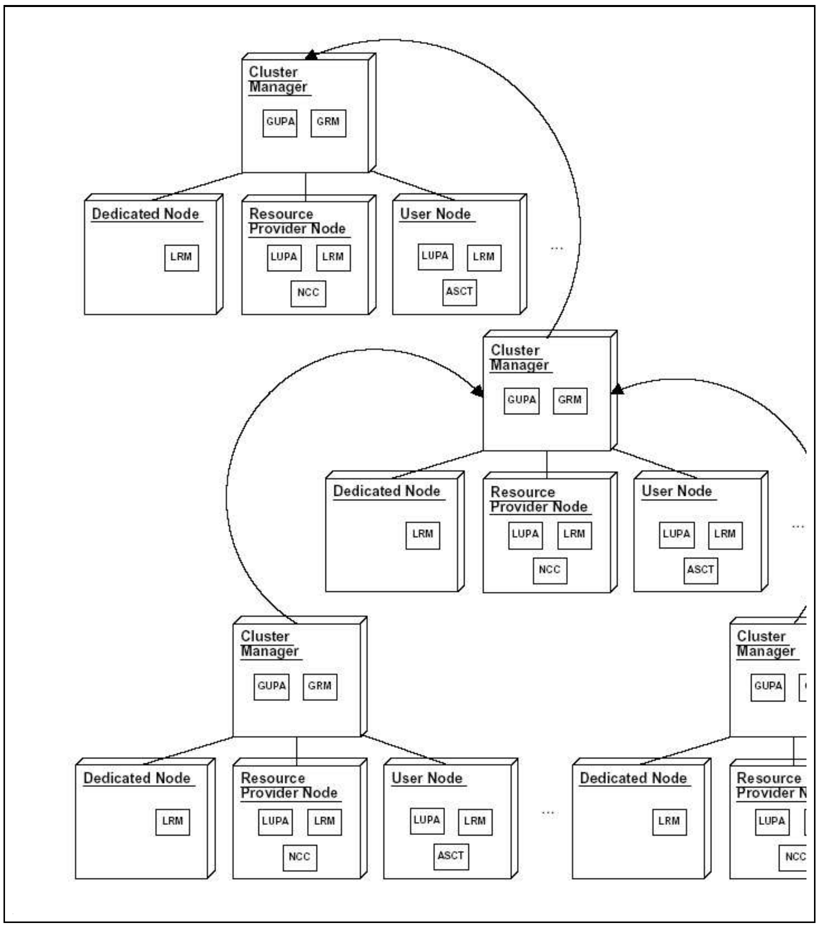

Figure 2.2: Integrade's Inter Cluster Architecture

vation and Execution Protocol, which works as follows: When a grid user submits an application for execution, the GRM selects candidate nodes for execution of the task on the basis of resource availability and application requirements. For this purpose, the GRM uses its local information relating to the cluster status as a hint for locating the nodes best suited to execute the respective application. Subsequently, the GRM engages in direct negotiation with the selected nodes to ensure that they actually have sufficient resources available for execution of the application at the given moment and, if possible, reserves the relevant resources in the target nodes. In the event that the resources are not available in a certain node, the GRM selects another candidate node and repeats the process.

Similarly to the LRM/GRM cooperation, the Local Usage Pattern Analyzer

(LUPA) and the Global Usage Pattern Analyzer (GUPA) handle intra-cluster usage pattern collection and analysis. The LUPA executes in each cluster node that is a user workstation and collects data about its user usage patterns. On the basis of long series of data thus obtained, it identifies usage patterns reflecting the use of the respective node throughout the week. Each node's usage pattern is periodically uploaded to the GUPA. This information is made available to the GRM, enabling it to take more appropriate scheduling decisions as a node's idle periods can be predicted on the basis of its usage patterns.

The Node Control Center (NCC) permits the owners of resource providing machines to set the conditions for resource sharing if they wish to do so. Parameters such as periods in which they do not want to share their resources, the portion of resources that may be used by grid applications (e.g. 30% of the CPU and 50% of its physical memory), or definitions as to when to consider their machine idle can be set using this tool. The Dynamic Soft Real Time Scheduler enforces the conditions defined by resource owners. The Application Submission and Control Tool (ASCT) allows InteGrade users to submit grid applications for execution. The user can specify execution prerequisites such as hardware and software platforms, resource requirements such as minimum memory requirements, and preferences such as priority to be given to execution on a faster rather than a slower CPU. Additionally, the user may also employ the tool to monitor application progress.

The information on the InteGrade project indicated above has been published by [16]. Unfortunately, further information relating to the project is only available in Portuguese and many questions such as, in particular, network-related issues remain open. For the time being, however, it appears unnecessary to spend more time on further investigating this particular project as the source code of InteGrade is currently still only available as version 0.1, its development status being 3 (alpha). Therefore, we may assume that the InteGrade project has so far not proceeded beyond the stage of a favourable design and a prototype and is therefore not eligible for implementation in the framework of the dIST project. In the course of the evaluation of InteGrade, however, we have come to the conclusion that an opportunity for execution of parallel applications in the framework of dIST represents a nice-to-have feature rather than a real necessity. Communication between the individual threads of a program would be excessively difficult -

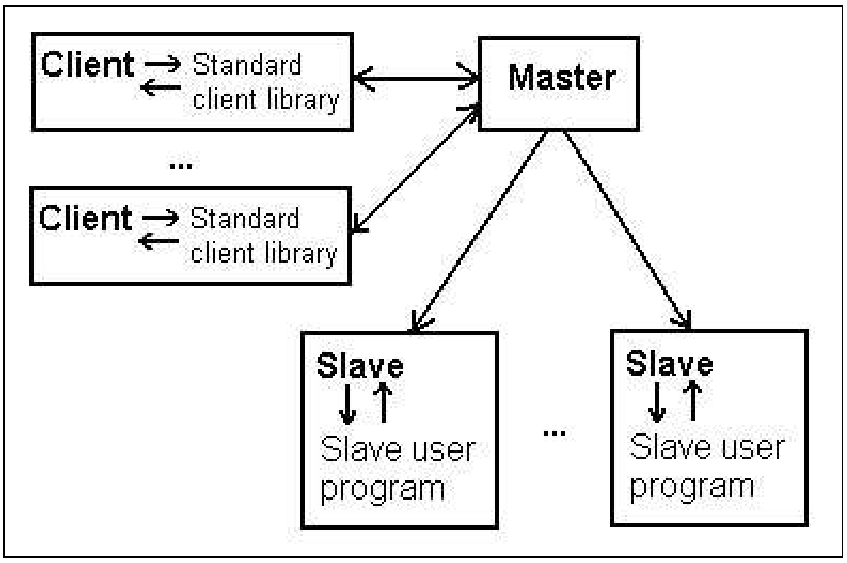

Figure 2.3: QADPZ architecture

and would require an excessive configuration effort that cannot be expected from the supporters of the dIST project - in a setting where clients are characterized by non-reliable availability and different quality of connection into the network and largely located behind routers and firewalls.

2.3 QADPZ

QADPZ, spoken "['kwod pi-'si]", stands for "Quite Advanced Distributed Parallel Zystem" [26] and is an open source implementation for distributed computing released under the General Public License (GPL) [13]. The system allows management/use of the computational power of idle computers in a network. The users of the system can transmit computing tasks to these computers for execution in the form of a dynamic library, an executable program or any program that can be interpreted (Java, Perl, etc.). Platforms supported are Linux, Unix, Win32 and MacOS X. The system is a client-master-slave architecture using message based communication. Messages between the components of the system are in XML format and can optionally be crypted for security reasons.

A QADPZ system consists of one master, many slaves and multiple clients delegating jobs to the master.

- Master - A process running on the master computer responsible for jobs-tasks-slaves accounting. The slaves talk to the master when they join or leave the system, or receive or finish tasks; the clients talk to the master when they start or control user jobs or tasks.
- Slave - A process running on the slave computer as a daemon or WinNT-service. The slave communicates with the master (possibly also directly with a client) and starts the slave user process. Without the slave running, the slave computer cannot take part in collaborative computing.
- Client - A process running on a client computer. It communicates with the master to start and control jobs and tasks of a specific user, may also communicate directly with slaves and is responsible for scheduling the tasks of user jobs as required by particular user applications (beyond the scope of scheduling done at the master level).
- Slave computer - One of many computers where the distributed collaborative computation takes place, for example: a UNIX server, a workstation, a computer in a student PC LAB, etc.
- Client computer - Any computer that the user uses to start his application. A client computer may be a notebook connected to the network using a dial-up connection, a computer in the office, lab, etc.

All slaves participating in the system are running a slave service program (a small resident application that accepts the tasks to be computed). The master is also running permanently and keeps track of the status of the slave computers, i.e. registers whether they are idle, busy computing a QADPZ task, or disabled because a user has logged in. When a user wishes to use the system, (s)he prepares a user application consisting of two parts: A slave user program, i.e. the code that will effect the desired computation after being distributed to the slaves, and the client that will be generating jobs to be computed.

The main role of the master (the central component of the whole system) is to maintain the current availability status of the slaves, and to start and control the tasks. A client doesn't communicate with the slaves directly but transmits all its requests to the master. Several user modes are available. To keep things simple, the user can choose the QADPZ standard client (qadpz_run), which allows him/her to set up and submit a job. The job description is saved into an XML-formatted project file and can be manually edited by more advanced users. Alternately, a user may want to write his or her own client application to have full control over the submission of tasks (for example in cases where the user must wait for the results of the computation of an initial series of tasks and subsequently, on the basis of these results, transmits either one or another group of tasks). It is also possible to directly write a slave library to speed up the execution. In that case, the slave service or daemon will not start a new process with the downloaded executable, but a dynamic shared library will be loaded by the slave service process (daemon/service).

The website [26] for the QADPZ project also comprises a discussion of safety issues: The current QADPZ security scheme is designed to protect the security of the computers in the network, i.e., a malicious hacker cannot submit an alien piece of code to be executed instead of a user computational task. However, this scheme doesn't protect the QADPZ user data. A malicious hacker can monitor (packet sniffing) or alter (IP spoofing) the data or control information arriving back to master or client nodes and thus:

- put the slave nodes out of operation;
- put the master node out of operation;
- modify the result data submitted by slaves;
- do any other kind of harm to the computational process.

QADPZ is a project with features very closely meeting the requirements of the dIST project. As the clients are entitled to transmit computing orders to the master and the master distributes these to the slaves, the system is very flexible and a user can quickly launch a try without needing the administrator of a central server. With a number of modifications, QAPDZ would certainly be an option for dIST. However, QADPZ is only available in the form of a version 0.8 (beta), which indicates that the software is still in the middle of its growth phase, so that it can be assumed that it is not sufficiently verified yet as far as safety issues are concerned. Additionally, QADPZ currently seems to be designed only for application within an in-house network. The website of the project, at least, contains no information describing any network functionality exceeding such in-house application. Another factor that must be taken into account is that the planned dIST computing network will largely consist of quasi-anonymous participants in a public, insecure network. As a basis for the dIST activity, I am therefore looking for a project that largely resembles the QADPZ system but is designed, in terms of its network features, for operation in the internet, strongly emphasizes reliable safety features and, if possible, already includes a workunit administration feature.

2.4 SETI@home

In the Puerto Rican port town Arecibo, one of the world's largest radio telescopes [25] with a reflector diameter of 305 meters receives approximately 35 gigabytes of data per day. The project SETI@home [27] of the University of California, Berkeley, [28] is dedicated to identifying, within the data received, signals from extra-terrestric life. As this project requires enormous quantities of computing capacity and no super computer is available for this purpose, SETI@home went looking for alternative methods of providing the required computing capacity. The overall data stream is divided in terms of both time and frequency resulting in individual files each comprising 0.25 MB. These files are transmitted via the internet to the computers participating in the project which execute the computing tasks and subsequently upload the results to one of the project servers. SETI@home is one of the most successful distributed computing projects ever. According to a paper published by the head of the project [6], SETI@home is currently running on more than one million computers, thus providing the project with a computing capacity of 60 teraFLOPS (trillion floating point operations per second), i.e. a computing power exceeding that of NEC's Earth Simulator (35.61 teraFLOPS/s), one of the world's fastest computers.

While SETI@home Classic and its middleware formed one monolithic program, new versions of SETI@home are based on BOINC, a general-purpose platform for distributed computing projects.

2.5 BOINC

BOINC (Berkeley Open Infrastructure for Network Computing) [5] is a platform for public-resource distributed computing. It is being developed at the U.C. Berkeley Spaces Sciences Laboratory by the same group that also has developed and operates SETI@home.

2.5.1 BOINC: General information

As far as its features are concerned, BOINC uses the requirements of the SETI project as a starting point. Additionally, however, it offers a strict distinction between middleware and scientific application and has been designed with a view to enabling other projects than SETI@home to use BOINC as a basis as well. These include, for example:

- ClimatePrediction.net [9], for long-term climate forecasts;
- Predictor@home [23], testing methods for protein folding forecasts;
- Einstein@home [12], dedicated to locating gravitation waves and pulsars;
- LHC@home [20], computing simulations for the construction of a new LHC particle accelerator at the CERN.

2.5.2 Licensing

BOINC has been released under the GNU Lesser General Public License (LGPL) [19] and is therefore free software. The GNU LGPL license ensures that the licensee is entitled to disseminate copies of the software (even charging a price if (s)he wishes to do so), to obtain the source code of the free software and to modify it. In return for the rights thus granted, each person who has modified and/or passes on any software thus obtained under the license is obliged to grant the recipient all rights (s)he was granted and must also ensure that the recipient may obtain the source code. Additionally, die LGPL protects the programmers of the free software by excluding any kind of warranty for the software.

While most of the GNU software today is published under the GNU GPL [13], the LGPL used for licensing of BOINC is frequently used for libraries (e.g. the GNU C-Library) to permit linking between such libraries and non-free software. The main difference between the GPL and the LGPL used by BOINC therefore consists of the fact that linking between a non-free software with a library licensed under the LGPL is subject to less strict conditions than linking with a library licensed under the GPL.

2.5.3 Architecture of a BOINC project

Figure 2.4 shows the structure of a typical BOINC project.

The project backend is project specific. It supplies the work units and handles the computational results.

The HTTP data servers , typically one ore more apache servers running on Linux, distribute the files. These servers must be able to handle CGI programs with POST commands. They need not be owned by the project although it is possible for the whole server complex to reside on one physical machine.

The MySQL database is a relational database which stores information about participants, workunits and results.

Utility programs and libraries are used by the project backend to manipulate the database.

The webinterface is automatically generated by one of the scripts of BOINC. It is used for the interaction between participants and the project respectively for the maintenance of the project by its developers. This webinterface can also run on the same apache server as the dataserver does.

The scheduling server is a small cgi program used for communication with the core clients.

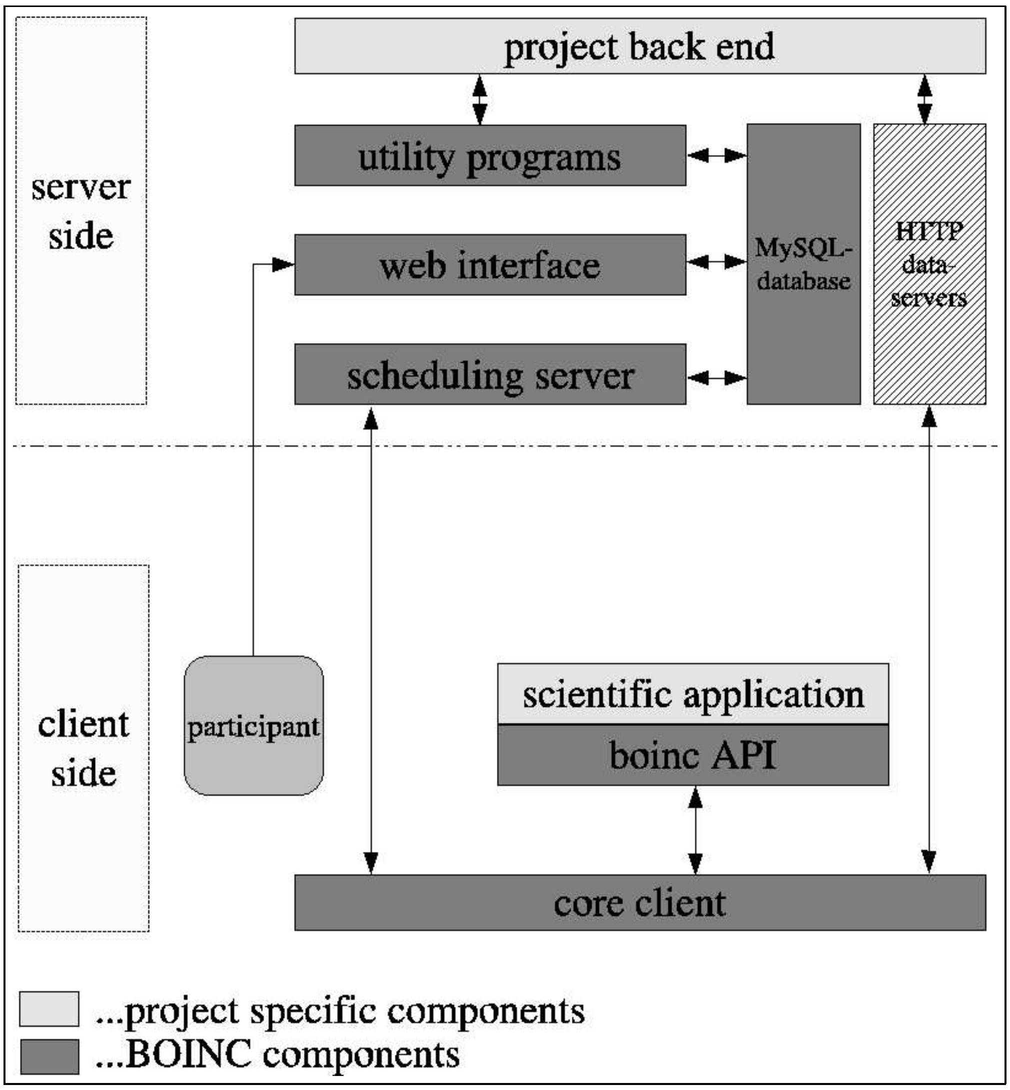

Figure 2.4: Structure of a boinc project

The core client is the base program running on the computers of the participants. It communicates with the serverside scheduling server and downloads the scientific applications and workunits from the data servers. The core client is responsible for starting the scientific application, for buffering its workunits and for uploading the results. The first time the core client is started it performs benchmarks and checks the number of cpu's and other parameters. This information is stored in the BOINC database and is later used to decide which workunits to transfer to the specific host. The base client is able to manage several scientific applications and projects. For example, if a participant decides to participate in Seti@home and in ClimatePrediction.net, one running instance of the core client manages both projects.

The scientific application is project specific. It has to be linked statically and also the BOINC API has to be linked with it. The BOINC API is used for communication between the scientific application and the core client. For example, system calls like the ones needed for file handling are performed using the API and also checkpointing is supported through the API. Because BOINC is released under the LGPL, the scientific application needs not to be open source software.

2.5.4 Selected features of BOINC

The users of course must not suffer any disadvantage as a consequence of participating in a BOINC project. Therefore, BOINC offers the participants an easy-to-operate webinterface they can also use to register for the project and to define their individual settings. Such settings include, for example:

- Do work while computer is running on batteries?
- Do work while computer is in use?
- Do work only after computer is idle for ...
- Do work only between the hours of ...

- Leave applications in memory while preempted?
- Confirm before connecting to Internet?
- Disconnect when done?
- Connect to network about every ...
- On multiprocessors, use at most ... processors

...and a lot of disk-, memory-, and bandwith-usage-limits. Since it is possible to have one account for multiple machines, such settings of one user account apply to all hosts attached to the account but the participants can also create separate sets of preferences for computers at home, work, and school. To keep participants contributing computing power BOINC provides an accounting system, where users get credits for doing calculations on workunits. It provides participant-oriented web site features such as

- the ability to form teams;
- the ability to create and browse "user profiles" including text and images;
- message boards, including a dynamic FAQ system that encourages participants to answer each others' questions.

According to [7] the experiance with Seti@home has shown that participants are highly motivated by credit, and particularly interested in their ranking relative to other users. This information is typically displayed on web-based leaderboards showing the ranking of participants or even teams of participants. For this matter, BOINC provides a mechanism that exports credit-related data in XML files that can be downloaded and processed by credit statistic sites operated by third parties. Several of these currently exist.

2.6 Conclusion

From all projects evaluated, BOINC with its architecture and features most properly meets the requirements of the dIST project. It can be assumed that BOINC, being widely disseminated, has been thoroughly tested and is sufficiently stable. Additionally, comprehensive documentation on the project as well as a number of very active mailing lists on the development and operation of BOINC are available. The software is licensed under the LGPL license, and modification and use of the software are therefore legitimate and free. Therefore, I have decided to use BOINC as middleware for application in the framework of the dIST project.

Chapter 3

The BOINC core client

A BOINC system consists of a server part and a client part. In the present chapter, the client part is described in terms of its structure, implementation and working method.

3.1 File structure

The core client runs in its home directory. It creates and uses the following files and directories:

`prefs.xml` contains the user's general preferences.

`client_state.xml` stores information about the host, the subscribed projects and their applications, workunits, results and similar information.

`account\_$<$PROJECT$>$.xml`, where the term PROJECT describes the URL of the project, contains the master URL, the user's authenticator and other project-specific preferences like the resource share rate which is used to determine how much of the host's idle time should be used for the described project. One such file exists for each subscribed project.

The directory `projects` contains a subdirectory for each subscribed project. This subdirectory contains all files (inputs, outputs, executeables) related to its project.

The directory `slots` contains one subdirectory, named 0, 1, ..., N-1, for each CPU, where the results execute.

3.2 FSM structure

Let us now consider a number of important details regarding the implementation of the core client. The core client consists of several finite-state-machines (FSMs):

NET_XFER: Each instance represents a network connection for which data is transferred to/from memory or a disk file.

HTTP_OP: Each instance represents an HTTP operation (GET, PUT or POST).

FILE_XFER: Each instance represents a file transfer in progress.

PERS_FILE_XFER: Each instance represents a "persistent file transfer", which recovers from server failures and disconnections and implements retry and give-up policies.

SCHEDULER_OP: There is only one instance. It encapsulates communication with scheduling servers, including backoff and retry policies.

ACTIVE_TASK: Each instance represents a running application.

FSMs of a particular type (one of the types explained above) are managed by an FSM container named after the type of the FSM with the suffix "`_set`". Each FSM container provides a (non blocking) `poll()` function for detecting and performing state transitions.

3.3 Data structures

The data types listed below represent the central data types of the core client, each class having write and parse functions for converting an instance to or from XML.

- PROJECT

- APP
- FILE_INFO
- APP_VERSION
- FILE_REF
- WORKUNIT
- RESULT

The initialization process works as follows: The "PREFS" class is a parsed version of the `prefs.xml` file. It contains a vector of partially-populated PROJECT objects and parsed versions of user preferences. When the core client starts up, the function CLIENT_STATE::init() is called. It parses `prefs.xml` and creates a "PREFS" object from the parsed data. If there is no prefs file in the home directory of the core client, it prompts the user for a master URL and account ID and creates one. Then the vector of "PROJECT" objects is copied to "CLIENT_STATE" which encapsulates the global variables of BOINC on this host and is a parsed version of `client_state.xml` represented as vectors of the basic types. CLIENT_STATE also includes transient variables such as sets of FSMs.

3.4 Main loop logic of the core client

The function CLIENT_STATE::do_something() is called repeatedly either by the sleep loop of the command-line program or by the time handler of the event loop in case the GUI program is used. This function initiates activities as needed and checks for the completion of current activities. If any change occured it returns true, in which case it should be called again immediately without sleeping. This is CLIENT_STATE::do_something():

```
bool CLIENT_STATE::do_something()
{
    bool action=false;
    if (check_suspend_activities()) return false;
    action |= net_xfers->poll();
    action |= http_ops->poll();
    action |= file_xfers->poll();
    action |= active_tasks->poll();
    action |= scheduler_rpcs->poll();
    action |= start_apps();
    action |= pers_xfers->poll();
    action |= handle_running_apps();
    action |= handle_pers_file_xfers();
    action |= garbage_collect();
    write_state_file_if_needed();
    return action;
}
```

Before the loop starts, `check_suspend_activities` checks for conditions in which user preferences dictate that no work be done (i.e. running on batteries). All the `poll()` functions manage the transitions of the various finite state machines. `start_apps()` checks whether it is possible to start an application and if so starts the application. The function `handle_running_apps()` checks whether a running application has exited, and if so cleans up after it. Then `handle_pers_file_xfers()` is called which starts and finishes file transfers as needed. `garbage_collect()` checks for objects that can be discarded. For example, if a file is non-sticky and is no longer referenced by any work units or results, both the FILE_INFO and the underlying file can be deleted. If a result has been completed and acknowledged, the RESULT object can be deleted. And finally `write_state_file_if_needed()` writes the file `client_state.xml` if any of the above function has set the flag `client_state_dirty`.

3.5 Host measuring performed by the client

Because the server distributes workunits according to the properties of the host, the core client measures the following aspects of each host and reports them in every scheduling RPC. Their values are stored in the projects database:

- CPU performance: Integer ops/sec, double-precision floating-point ops/sec, and memory bandwidth are measured separately. They are measured by a process executing at the same priority as BOINC applications, so the results will be affected by other processes. These measurements are taken when the client starts for the first time, and once every month afterwards.
- Number of CPUs: By default, the number of simultaneous slot directories will be set to this number unless otherwise indicated by user preferences.
- Vendor and model of CPU
- Disk space: Free space and total space on the drive where BOINC is installed. These numbers will be used to prevent BOINC from using more space than set in the user preferences.
- Memory: Total RAM, CPU cache, and swap space. These numbers can be used by the scheduling server to decide whether or not to assign work to a client. This also provides a means for assignment of differing work based on host abilities.
- Timezone
- Last IP address and count of consecutive same addresses.
- Number of RPCs, and time of last RPC.
- Fractions of time that core client runs on host, host is connected, and user is active. These are computed as exponentially-weighted averages.
- Operating system name and version.
- Average up- and downstream network bandwidth. These are computed as exponentially-weighted averages.

3.6 Client's CPU scheduling policy

CPU scheduling means that the CPU time at BOINC's disposal is appropriately used for the host's projects. For example, a user may decide to dedicate 75 percent of his/her spare computing time to one project and 25 percent to another. In this case, there will be one client running on his/her system and the value `resource_share`, which is part of the preferences, determines how much time will be granted to each project. In this context, the variable "`debt`", which is used for making scheduling decisions, must be introduced: For each project, `debt` means the amount of CPU time the BOINC client owes to the respective project. `debt` decreases while calculations are performed for the project and increases according to the total amount of work done in a time period scaled by the value `resource_share`. To illustrate what this means, let us take another look at the example referred to above: Let us assume that the user participates in both CAPE (see Section 7.5) and the HappyEnd project (see Section 7.2):

- Project CAPE: `resource_share=75`
- Project HappyEnd: `resource_share=25`

Further suppose that the system has 40 minutes of spare computing time left and the BOINC client runs CAPE for 25 minutes and HappyEnd for 15 minutes. Because the `resource_share` of CAPE is 75, CAPE was entitled to be allocated 30 minutes but got only 25. Therefore its `debt` must be increased by 5 minutes (i.e. decreased by 25 minutes but increased by the anticipated 30 minutes).

Actually, the algorithm computes the "anticipated debt" (the `debt` expected to be owing to the project after expiry of the time period) for determining which computation to start next. The ratio between the CPU times allocated to the projects in the course of one or two days should approximately match the ratio between the user-specified resource shares. By the way: There is no need for the sum of all `resource_share` rates to be exactly 100.

If there are no workunits waiting to be computed for a particular project, its `resource_share` is of course not taken into account for the purpose of determination of the debts for other projects, and its own debt is not increased. Thus the debt for such a project is set to zero. [5] provides a pseudo code scheduling algorithm and states that the clients CPU scheduling algorithm aims to achieve the following goals in decreasing priority:

1. Maximize CPU utilization.
2. Enforce resource shares.
3. Satisfy result deadlines if possible.
4. Reschedule CPUs periodically. This goal stems from the large differences in duration of results from different projects. Participants in multiple projects will expect to see their computers do work on each of these projects in a reasonable time period.
5. Minimize mean time to completion. For example, it is better to have one result from a project complete in time T than to have two results simultaneously complete in time 2T.

The results to compute are dynamically chosen from a global pool so as to minimize the number of result computations active at any given time, and when CPU time is allocated to a project, already running tasks are chosen before preempted tasks and new result computations are only launched when no other tasks are waiting to be completed. The algorithm is as follows:

- Whenever a CPU is free
- Whenever a new result arrives (via scheduler RPC)
- Whenever it hasn't run for T seconds, for some scheduling period T

```
if (a project has no runnable results)
{
  Reset the projects debt to 0;
  do not consider its resource share to determine relative
   resource shares;
}
else
```

```
{
  Decrease debts to projects according to the amount of work
   done for the projects in the last period.
  Increase debts to projects according to the projects
   relative resource shares.
}
for (each project)
{
  anticipated debt = current debt
}
while(a result to compute is not determined for every cpu):
{
  Choose the project that has the largest anticipated debt
   and a ready-to-compute result.
  Decrease the anticipated debt to the project by the expected
   amount of CPU time.
}
Preempt current result computations, and start new ones.
```

This piece of pseudocode describes the algorithm more closely:

```
data structures:
ACTIVE_TASK:
    double cpu_time_at_last_sched
    double current_cpu_time
    scheduler_state:
        PREEMPTED
        RUNNING
    next_scheduler_state    // temp
PROJECT:
    double work_done_this_period    // temp
    double debt
    double anticipated_debt // temp
    RESULT next_runnable_result
```

```
schedule_cpus():

foreach project P
    P.work_done_this_period = 0

total_work_done_this_period = 0
foreach task T that is RUNNING:
    x = T.current_cpu_time - T.cpu_time_at_last_sched
    T.project.work_done_this_period += x
    total_work_done_this_period += x

foreach P in projects:
    if P has a runnable result:
        adjusted_total_resource_share += P.
        resource_share

foreach P in projects:
    if P has no runnable result:
        P.debt = 0
    else:
        P.debt += (P.resource_share /
        adjusted_total_resource_share)
                * total_work_done_this_period
                - P.work_done_this_period

expected_pay_off = total_work_done_this_period /
num_cpus

foreach P in projects:
    P.anticipated_debt = P.debt

foreach task T
```

```
    T.next_scheduler_state = PREEMPTED

do num_cpus times:
    // choose the project with the largest anticipated
    debt
    P = argmax { P.anticipated_debt } over all P in
    projects with
         runnable result
    if none:
        break
    if (some T (not already scheduled to run) for P is
    RUNNING):
        T.next_scheduler_state = RUNNING
        P.anticipated_debt -= expected_pay_off
        continue
    if (some T (not already scheduled to run) for P is
    PREEMPTED):
        T.next_scheduler_state = RUNNING
        P.anticipated_debt -= expected_pay_off
        continue
    if (some R in results is for P, not active, and
    ready to run):
        Choose R with the earliest deadline
        T = new ACTIVE_TASK for R
        T.next_scheduler_state = RUNNING
        P.anticipated_debt -= expected_pay_off

foreach task T
    if (scheduler_state == PREEMPTED and
    next_scheduler_state
         = RUNNING) unsuspend or run
    if (scheduler_state == RUNNING and
    next_scheduler_state
```

```
            = PREEMPTED) suspend (or kill)

foreach task T
    T.cpu_time_at_last_sched = T.current_cpu_time
```

3.7 Client's work fetch policy

Clients need not necessarily be connected to the server day and night. Therefore, an algorithm is implemented to fetch enough work to avoid starvation for a period of T days, starvation meaning that the CPU scheduler chooses a project to run for which no computation tasks are scheduled. On the other hand, the volume of work fetched should not be greater than necessary to achieve this goal. Therefore, the algorithm tries to maintain enough work for T to 2T days.

To meet these requirements, the algorithm takes various system and project parameters into account such as the number of CPUs, the `resource_share` rates of the projects, the active fraction (which means the fraction of time in which the core client typically runs), the CPU speed and the `estimated_cpu_time` for a result.

Chapter 4

The BOINC server

4.1 BOINC server - general information

From a hardware point of view, a BOINC server may be an inexpensive standard PC set up with a LAMP system, the abbreviation LAMP standing for Linux-Apache-MySQL-PHP. The all-in-one server used for the dIST project, for example, is a Pentium-4 machine with 3.0 GHz and 1 GB RAM as well as a RAID system consisting of 6 ATA disks with 200 GB each, running Debian Woody with Linux kernel 2.6.7. The installation and configuration of BOINC on this server will be described in Chapter 6. In addition to the LAMP system mentioned above, which only represents the basic installation, the server consists of a database and a project directory for each single project it runs. The project directory contains the subdirectories and files listed below:

- `apps/forapplications` ready to be added to the system
- `bin/` contains scripts and programs to be described later on
- `cgi-bin/` contains cgi programs called by the clients via the apache webserver
- `config.xml` is a file with general settings such as paths, URLs, etc.
- `download/` for downloading of the core client and the scientific applications

- `html/` provides a complete project website including a variety of php programs
- `log_$<$hostname$>$/` contains log files for the programs running
- `pid_$<$hostname$>$/` contains small files with the process ID's of the `feeder`, the `file_deleter` and the `transistioner` as well as lock files for those programs
- `project.xml` describes supported platforms
- `upload/` is the directory where the clients upload their results

4.2 Project database

Each project has its own MySQL database. This database consists of various tables, the most important being:

platform, a table containing platforms designed as follows:

ID	CREATE_TIME	NAME	USER_FRIENDLY_ NAME	DEPRE- CATED
1	1100212499	i686-pc-linux-gnu	Linux/x86	0
2	1100212499	windows-intelx86	Windows/x86	0

app, a table where application information is stored.

app_version, a table where the different versions of an application are stored together with information on the appropriate BOINC core client versions and platforms designed as follows:

ID	CREATE_ TIME	APP ID	VER SION_ NUM	PLAT- FORM ID	XML_ DOC DOC	MIN_ CORE_ VER- SION	MAX_ CORE_ VER- SION	DE- PRE- CATED
1	1098876459	1	402	1	BLOB	0	0	0

user contains the names of the users, their email addresses, authenticators (32 bit hash values), locations and a number of user-specific preferences.

host stores host-specific information such as domain_name, last_ip_addr, vendor of the processor, processor model, operating system, etc.

workunit describes workunits. The input file descriptions are stored in an XML document in a blob field. The table includes counts of the number of results linked to the respective workunit, as well as of the numbers of workunits that have been sent, that have succeeded and that have failed. A more detailed description of the workunit table is included in section 4.3.

result describes results. It includes a "state" and stores items relevant only after the result has been returned: CPU time, exit status, and validation status. A more detailed description of the result table can be found in section 4.4.

4.3 Workunit parameters

One row in the workunit table of a project database describes the attributes of one workunit representing one computation task to be performed. A number of tools, described in section 6.5, is available for creating workunits. In the present section, the different attributes are to be explained.

4.3.1 General properties

name is the name of the workunit and must be unique.

appid is the ID of the application to perform the computation.

input files is a list of the input files for the computation.

priority means that higher-priority work is dispatched first.

batch is an integer field that can be used to operate (cancel, change priority, etc.) on groups of workunits. Unfortunately there is no more information about the usage of this value available in the documentation of Boinc [5].

4.3.2 Resource estimates and bounds

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

rsc_fpops_bound: If the number of floating-point operations required to complete a computation exceeds this bound, the computation is aborted.

rsc_mem_bound is a bound on the virtual memory working set size, and a workunit is only sent to a host if its rsc_mem_bound value does not exceed the host's capabilities. If an application exceeds this bound, it is aborted.

rsc_disk_bound is a bound concerning disk space that works like rsc_mem_bound.

4.4 Result parameters

Deviating from the common definition, the term "result" in a BOINC context does not refer to the final result of a computation process but to the instance of a computation irrespective of whether the respective computation is yet unlaunched, in progress or completed. Project databases contain a "result" table where the results are stored. BOINC automatically creates these entries for workunits on the basis of their redundancy parameters. The parameters of a result include:

name, i.e. the unique name of the result derived from the name of the associated workunit.

workunitid, i.e. the ID of the associated workunit.

xml_doc_in, i.e. a list of the names of the output files and the names by which the application refers to them.

server state, i.e. a dynamic attribute. The following states are encoded in integer values:

- Inactive - not ready to dispatch
- Unsent - ready, but not sent yet

- In progress - sent but not yet done
- Done successfully
- Timed out
- Done with error
- Not needed - the work unit was finalized before this result was sent

hostid, identifying the host that executed the computation.

exit_status, representing what its name implies, 0 meaning success.

CPU_time, indicating the computation time of the result.

xml_doc_out, describing the output files, their sizes and checksums.

stderr_out, being the stderr output of the computation.

received_time, indicating the time when the result was received.

4.4.1 Redundancy and scheduling

delay_bound is the maximum number of seconds allowed to elapse between sending a result to a client and receiving a reply. The scheduler will not issue a result if the estimated completion time exceeds this bound. If the client does not respond within the time permitted, the server cancels the result and generates a new one to be reassigned to another client. This value should be set to several times the estimated average execution time of the workunit on an average PC. If it is set too low, no results can be sent and the corresponding workunit is flagged with an error.

min_quorum is used to achieve redundant computing ($min\_quorum > 1$).

target_nresults specifies how many results are to be initially created. The lower bound for this value of course is *min_quorum*, and the value may be set higher to compensate for result loss or to obtain results more quickly.

max_error_results indicates that a workunit is to be considered erroneous in the event that the number of errors occured exceeds the value defined. In this case, no further results are issued. This feature functions as a safeguard against upload/download problems and client crashes.

max_total_results specifies that a workunit is to be considered erroneous if the total number of results for the workunit exceeds the value defined. This feature functions as a safeguard against client crashes.

max_success_results is the maximum number of successful results permitted to be returned without consensus. This feature functions as a safeguard against workunits that produce nondeterministic results.

4.4.2 Error mask

As mentioned above, workunits may be classified as erroneous. For this case, the database provides an error mask, which is a bit mask of error conditions:

- WU_ERROR_COULDNT_SEND_RESULT: Either no application version was available for the platform of the hosts or a large number of hosts failed to meet the resource requirements as the exceeded the disk, memory or CPU limits described above.
- WU_ERROR_TOO_MANY_ERROR_RESULTS: Too many results with error conditions have been returned for the respective workunit.
- WU_ERROR_TOO_MANY_SUCCESS_RESULTS: Too many successful results have been returned without consensus.
- WU_ERROR_TOO_MANY_TOTAL_RESULTS: Too many total results have been sent for the respective workunit.

If any of these conditions holds, BOINC ”gives up” on the workunit and refrains from dispatching any more results for it.

4.5 Daemon programs

Every BOINC project has a set of daemon programs used to manage the work and maintain the project. These are listed within the daemons section of the xml-file `config.xml` in the project directory. These daemons include:

- Work Generator
- Transitioner
- Validator
- Assimilator
- file_deleter
- db_purge

4.5.1 The Work Generator

The Work Generator is a project-specific program the purpose of which consists of generating workunits with appropriate input files if needed. This may be a simple shell script using the `create_work` program or a program using the C++ libraries `crypt.C` and `backend\_lib.C,h`, which provide the following functions:

- `int create_work(...)`, which creates a workunit and one or more results.
- `read_key_file()`, which reads the file upload authentication key.

The process of work generation works as follows:

1. A workunit template file must exist. It 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_bounds> ]
    [ <rsc_disk_bound>x</rsc_disk_bounds> ]
    [ <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>
    ]
</workunit>
```

The parameters of this file are described in section 4.3.

2. A result template file which 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>
```

...where `<OUTFILE_n>` is replaced by a string of the form "`<workunitname>_resultnum_n`", where `resultnum` is the ordinal number of the result (0, 1, ...). `<UPLOAD_URL/>`, is replaced with the upload URL specified within `config.xml`.

3. The input files for the workunit must be placed in the download directory.
4. Either the `create_work` utility or the library function mentioned above must be called to create the workunit within the project database where the entry `transition_time` of the workunit table is set to the current time.

4.5.2 The Transitioner

The Transitioner is part of the BOINC software and independent of the project. It is listed in the `config.xml` file by default. The Transitioner is run whenever a result becomes done, the error mask of a workunit is set or assimilation is finished. It handles state transitions of workunits and results, generates initial results and is responsible for generating more results when timeouts or errors occur.

4.5.3 The Validator

The Validator is another project-specific feature. It is used to compare redundant results and to select a canonical result as the correct one. In this respect, care must be taken when comparing results because floating-point arithmetic varies between different platforms. Additionally the Validator is responsible for credit granting. BOINC supplies a framework program `validator.C`, which must be linked with two application specific functions to make a validator program:

- function `int check_set(vector<RESULT> results, DB_WORKUNIT& wu, int& canonicalid, double& credit, bool& retry);`
  - Compares the output files of a set of results and if there is a quorum of matching results, it selects one as the canonical result, returning its

ID. Credit is granted to the users who return correct results.

- Sets the result's outcome (in memory, not database) to outcome=RESULT_OUTCOME_VALIDATE_ERROR and validate_state=VALIDATE_STATE_INVALID if an output file for a result has a non-recoverable error.
- Sets the `validate_state` field of each non-ERROR result (in memory, not database) to either `validate_state`=VALIDATE_STATE_VALID or `validate\_state`=VALIDATE_STATE_INVALID.
- Returns retry=true in case of a recoverable error while reading output files to instruct the Validator to process this workunit again later.
- Returns non-zero if a major error occurs to tell the validator to exit.

- function `int check_pair(RESULT& new_result, RESULT& can_result, bool& retry);`
  - Compares a new result to the canonical result and sets the new result's validate state to either VALIDATE_STATE_INVALID or VALIDATE_STATE_VALID.
  - Sets the new result's outcome (in memory, not database) to VALIDATE_ERROR if a nonrecoverable error occurs reading the output file of either result or the new result's output file is invalid.
  - Returns `retry=true` if it has a recoverable error while reading an output file of either result.
  - Returns nonzero if a major error occurs.

BOINC provides a placeholder, `validator_placeholder.C`, which can be used for writing a project specific Validator. A Validator has the following command-line arguments:

- `-app appname`: The name of the application
- `-one_pass_N_WU N`: Validate at most N Workunits, then exit
- `-one_pass`: Make one pass through WU table, then exit

- `-mod n i`: Process only workunits with $(id\ mod\ n) == i$. This option can be used to run multiple instances of the validator for increased performance.

Chapter 5

Modifications

This chapter describes the areas where the BOINC functionalities were found insufficient for the purposes of the dIST project as well as the modifications and extensions added to the original BOINC.

5.1 Motivation underlying the modifications

The original version of the system works as follows: One or more projects are located on the server and each project has its own webinterface. A participant registers to one of the projects via its webinterface. Following this, a 32 bit key unique within the project is sent to the participant's email address. The participant then uses the 32 bit key to confirm the account via the webinterface. This step certainly being required from a safety point of view to prevent DOS attacks resulting from automatic account generation. Following this, the participant launches the client and enters the master URL and the 32 bit key (s)he has received. The same procedure is repeated for each of the projects the participant wishes to support.

In contrast to this arrangement, the dIST project is intended to procure computing time for a variety of different projects in general, the projects dynamically evolving and terminating from time to time. For this purpose, an arrangement where the participants must be contacted for each new project so that they can rearrange their clients will not be acceptable, and the administrators in charge of the computer rooms of the TUG can certainly not be expected to separately change the settings of each individual computer whenever a change occurs. Additionally, the value `resource_share` is to be administrated by the dIST project so that the priority allocated to each of the individual projects can be managed on the level of the server. This type of arrangement is, among other things, intended to offer an opportunity to reduce the priority of a long-term project on a temporary basis while another, more important project is computed for a few weeks. Additionally, a special simplified registration feature that is described in Section 5.3 is to be made available for special cases.

5.2 Automatic participation in projects

In the original version, the server can be used to run several scientific projects at the same time. The modified BOINC version, in contrast, will additionally comprise a special project, i.e. the administrative project called "dIST", which will not distribute workunits but contain, on the server side, the extended functionality consisting of

- automatic participation in scientific projects, and
- simplified registration for the administrative project.

The users are intended to use the extended BOINC client. This client already knows the URL of the administrative project and, to get launched, only needs the 32 bit key. During start-up and at regular intervals thereafter, the modified client connects to the server or, more precisely, the script `automatic_subscription.php` of the administrative project. This script evaluates the file `administrative_config.xml`, which is also located on the server, and checks all project databases to determine if the respective user is already registered there. If this is not the case, the script registers the user in the databases of the respective scientific projects and sends a list containing the parameters of the projects to be supported to the client. This list is stored in the form of an XML file in the work directory of the client under the file name `auto_proj_list.xml`. The file `administrative_config.xml` is designed as follows:

```
# The XML parser of BOINC is very simple. Therefore
  some rules:
#
# 1) TRUE and FALSE are expressed by 1 and 0
# 2) The order must be project0, project1, ...,
  projectn
#    do not omit numbers!
# 3) No comments within the tag section. Just to be
  sure...
# 4) The master_url is the first line of every project
#    section WITH "http://"-prefix
#
# It is essential that this file is absolutely correct
# because the clients do not check every special case
  that
# can occur and they could choke on a wrong syntax.

<administrative_config>

<reset_everything>0</reset_everything>

<project0>
 <master_url>http://fsmtdist.ist.tu-graz.ac.at/coffee</
   master_url>
 <resource_share>500</resource_share>
 <db_name>coffee</db_name>
 <detach>0</detach>
</project0>

<project1>
 <master_url>http://fsmtdist.ist.tu-graz.ac.at/hello</
   master_url>
 <resource_share>8</resource_share>
 <db_name>hello</db_name>
 <detach>0</detach>
</project1>

</administrative_config>
```

Any new scientific project generated by the operators of the BOINC server can be set up in strict compliance with the standard without taking the BOINC extensions into account. In the event that the new project is to be supported by the clients of the administrative project dIST after the test phase, it only needs to be entered in the file `administrative_config.xml` according to the rules outlined above. The new project is automatically supported as soon as the clients, lodging queries with the server at regular intervals, detect it. Caution is recommended, however: As already explained in the file above, the structure of the XML parser included with the regular BOINC distribution is very simple. Consequently, the error tolerance of the extended version using the same parser is rather limited and the file must be 100% correct. Explanation of the tags of `administrative_config.xml`:

`<reset_everything>` specifies complete deletion on the client side of all projects supported at a given time (i.e. including the scientific applications, data, etc.) and subsequent subscription of the projects contained in the list. This tag should, as a matter of fact, never be needed but has been included for reasons of safety so that a clean-up can, if required, be performed on the client side. The administrative project is not covered by this tag.

`<resource_share>` indicates the priority of the project, therefore gaining importance as soon as more than one project is active. An exact description of the way this value is used for CPU scheduling purposes can be found in Section 3.6.

`<detach>` indicates that an individual project need no longer be supported. The clients fully delete the applications and data of the respective project.

5.3 Registration feature

The original BOINC version requires the user to create an account via the webinterface. Following this, a 32 bit key is sent to the user's email address. The user then enters the key via the webinterface and must finally indicate the master URL as well as the key when initially launching the BOINC client. While it definitely makes sense and increases safety, this procedure can in certain cases be rather troublesome. In a situation where several accounts are to be created, for example, a separate email address would be required each time. This is because the email address must be unique. Additionally, setup is to be facilitated for the administrators within the in-house network. Therefore, I have included, for such cases and in addition to the regular registration, a simplified and more comfortable registration feature. For this purpose, the client, upon initially launching, is offered the option "-auto_acc" followed by the parameters for the registration:

```
./boinc_client - <username> <mail> <country> <postal
  code> <secret>
```

Example:

```
./boinc_client -auto_acc ''Bernhard Kornberger''
  kornberg@sbox.tugraz.at Austria 8010 theKey
```

A parameter containing spaces must of course be put in quotation marks as the shell will otherwise interpret it as two separate parameters. Following this, the client registers for the administrative project dIST with these parameters. From this moment, the registration data is stored, and the client can in the future be called using `./boinc_client`.

A client started this way transmits an inquiry to the server via HTTP, the recipient of the inquiry being the script `automatic_account_generation.php` within the administrative project. This script verifies that the secret key `<secret >` is correct and whether or not automatic account generation is permitted at the given moment. Subsequently, the script generates an account with the administrative project and transmits a reply containing the account data to the client. This reply runs as follows:

```
# The account informations below were generated
  automatically using AustroBOINC
# and apply to this machine only:

<user>Bernhard Kornberger_auto-1</user>
<email>kornberg@sbox.tugraz.at</email>
<email_stored_in_db>kornberg@sbox.tugraz.at</
  email_stored_in_db>
<administrative_url>fsmtdist.ist.tu-graz.ac.at/dist</
  administrative_url>
<authenticator>ac59f81386aaeccb064a1b285d003db9</
  authenticator>

# Thank you for contributing computing power to this
  project!
```

...and is stored in the form of a file called `auto_admin_proj_acc.xml` on the client's hard disk. The following reply would be transmitted for a second account generated with exactly the same data:

```
# The account informations below were generated
  automatically using AustroBOINC
# and apply to this machine only:

<user>Bernhard Kornberger_auto-2</user>
<email>kornberg@sbox.tugraz.at</email>
<email_stored_in_db>kornberg@sbox.tugraz.at_duplicate
  -2</email_stored_in_db>
<administrative_url>fsmtdist.ist.tu-graz.ac.at/dist</
  administrative_url>
<authenticator>03ecd3133453727350a23098cb8f3691</
  authenticator>
```

```
# Thank you for contributing computing power to this
  project!
```

As shown by these replies, accounts generated automatically are always designated by the suffix "`_auto-n`" added to the name. To meet the BOINC requirement of each email address being registered only once in the database, the suffix "`_duplicate-n`" is added to already existing email addresses. It is fairly obvious that the simplified registration feature gives rise to a risk of DOS attacks as a third party could automatically create millions of accounts so as to overload the server. This risk is counteracted by two measures:

1. A secret key <secret> is required to use the feature. This key is only known to authorized persons. This key is, on the server level, defined directly in the file "automatic_account_generation.php".
2. The tag <disable_account_creation/> can be entered in the file `config.xml` so as to deactivate the function. Automatic registration will mainly be useful during the initial phase of the project and can then be blocked using this tag.

5.4 Implementation details

Implementation is based on the code boinc-cvs-2004-09-08.tar.gz, the extensions being implemented as far as possible in the two new files `austroBoinc_utils.h` and `austroBoinc\_utils.C` so as to facilitate upgrades. However, programming work will be required in the event that the client should in the future be based on a different source code. The full functionality on the server level is exclusively implemented in PHP scripts and XML files so that upgrades on the server level will not require a significant effort. Additionally, the extensions on the server level are limited to the administrative project so that new projects can be established without taking the modifications into account. A CD-ROM containing both the source code `boinc-cvs-2004-09-08.tar.gz` and the extensions is annexed to the present diploma thesis so that the setup of the server extended by the functions described can easily be repeated. The application of the new function is documented in Section 6.7.

Chapter 6

Setup and usage

This chapter describes in full how to set up a BOINC server. Setting up our dIST server for the first time has made us aware of a great number of obstacles on the way to a fully working BOINC system. Therefore, this chapter contains a considerable number of notes I have jotted down during the setup procedure. These notes represent a valuable part of the present diploma thesis as they ensure that others will be able to follow the processes described, and maybe they will even help other developers on their way into the world of BOINC.

6.1 System setup

This section explains how the basic system must be configured so as to allow running of BOINC.

6.1.1 Linux distribution

In principle, almost any desired Linux distribution may be used. However, BOINC is developed on Debian stable/unstable, Red Hat 8 and Solaris 2.6-2.9, and therefore it can be assumed that use of these distributions will reduce problems to a minimum and give best access to support. Initially, I attempted to install BOINC on Mandrake 10.0, which for not fully clarified reasons led to shared memory problems so that the setup was not successful. A second try on Debian Woody (Kernel 2.4.20-bf2.4) was successful.

6.1.2 Installation of additional packages

The BOINC documentation lists a number of packages that must be subsequently installed on Debian systems, but this list is of course not exhaustive for lack of a common basic installation. After installing Debian Woody, we had to install the following additional packages on the dIST server:

```
apt-get install g++ python python-mysqldb python-xml
mysql-server mysql-client apache php4 automake autoconf
php4-mysql python2.2-xmlbase python2.2-mysqldb
python2.2 libmysqlclient10-dev zlibc zlib1g zlib1g-dev
```

These packages can be obtained from several standard Debian repositories.

6.1.3 Python update

As indicated in the paragraph above, we subsequently installed Python 2.2. The command ”python -V”, however, showed the old Python version 2.1.3, which had obviously not been removed in the course of the installation process. Therefore, we had to insert a new symlink manually:

```
[root@wtdist /usr/bin]# rm python
[root@wtdist /usr/bin]# ln -s python2.2 python
```

6.1.4 httpd.conf

The file `/etc/apache/httpd.conf` is the configuration file of the Apache web-server. In this file, a number of changes must be made:

- In the course of installation of the packages indicated above, one of the installation scripts offered to enter the line

  ```
  LoadModule php4_module /usr/lib/apache/1.3/libphp4.
  so
  ```

  into the configuration file and after completion of the installation process the line actually appears in the `httpd.conf` but is commented and must be subsequently activated.

- To activate PHP, the following lines must be uncommented:

  ```
  AddType application/x-httpd-php .php
  AddType application/x-httpd-php-source .phps
  ```

- The DirectoryIndex directive controls which page will be shown when a directory is called by a web browser. This line already exists in `httpd.conf`, but it has to be extended by ”index.php”:

  ```
  DirectoryIndex index.html index.htm index.shtml
  index.cgi index.php
  ```

6.1.5 MySQL

For security reasons, accessibility of MySQL-Server is to be limited to the local level. Therefore, it is protected as follows:

```
/etc/mysql/my.cnf:
# prevent access from the network ...
# ... but listen to the loopback-dev:
bind-address     = 127.0.0.1
#skip-networking
```

To nevertheless allow remote administration, the package "phpmyadmin" can be installed. This package offers access to the MySQL database of BOINC via a comfortable web interface. Then the root password is defined as follows:

```
mysqladmin -u root password [new-password]
mysqladmin -u root -p reload
```

6.1.6 Sending of e-mails

Registration of new users is implemented in the BOINC webinterface by means of PHP scripts. These must be able to send confirmation e-mails containing a 32 bit key to the new users. In a first attempt, I tried to configure sendmail for this purpose. Safely configuring and operating a true mail server, however, is a delicate task. Additionally, port 25 of the network of the University of Technology Graz seems to be locked for precisely this reason. For reasons of simplicity and security, installing SSMTP represented a significantly more favourable alternative in comparison with sendmal, SSMTP being a Mail Transfer Agent (MTA) to get mail off the system to a mail hub. To cause SSMTP to be actually used, the following entry is required in the file `''/etc/php4/apache/php.ini''`:

```
sendmail_path = /usr/sbin/ssmtp -t -i
```

The path `''/usr/sbin/sendmail''`, in fact consisting of a symlink on SSMTP, turned out to be insufficient as a parameter, i.e. the options "-t" und "-i" must be explicitly indicated even though they are indicated as default options in the documentation.

6.2 Setting up BOINC

After completing the basic configuration of the LAMP server (section 6.1), installation of BOINC as such can be commenced.

6.2.1 Selecting the right source code

The development of BOINC is ongoing, and therefore the relevant documentation is also subject to frequent changes. Therefore, attention is expressly called to the fact that the information below is based on the situation as of summer 2004. Selecting the right BOINC source code is a critical issue. The website `http://boinc.berkeley.edu/source` offers many different versions of the source code but provides no information as to which of the source codes is considered stable and should be used for servers. The website comprises two download directories, i.e. "Archive" and "Nightly".

- The following facts must be taken into account with regard to the directory "Archive":
  - The 2.x versions can be compiled but are obsolete.
  - Many 3.x and 4.x versions are defective. Some of the versions do not contain some of the files needed for compiling. In some other cases, the test scripts indicate errors.
- The following information refers to the directory "Nightly":
  - The directory contains two different source variants: The "boinc-cvs-<date>" versions and the "boinc_public-cvs-<date>"versions.
  - Even though the public versions seem to contain all necessary components, they do not compile faultlessly, and the test scripts do not work either. In reply to a question I posted on a mailing list, a developer explained that the "boinc_public-cvs-<date>"are stable but only intended for the client while the development version "boinc-cvs-<date>" should be used for the server.

Even with the information above, selecting the source code was a critical task as CVS snapshots are made and provided for download on a daily basis and some of these snapshots simply do not compile. The source code `boinc-cvs-2004-09-08.tar.gz`, at any rate, works properly.

Update: Only a few days before finishing this diploma thesis, I received new information relating to the source code, i.e. that the development branch can be checked out using the command: `cvs -d :pserver:anonymous@alien.ssl.berkeley.edu:/home/cvs/cvsroot checkout boinc` *and subsequently the command* `cvs update -r boinc_core_release_4_66` *to get a source code which is known to build correctly (April 2005). For details see [24]. However, as our source code builds cleanly we will not upgrade our project.*

6.2.2 Downloading and compiling

Downloading and compiling of BOINC is very similar to downloading and compiling of typical Linux installations. The only additional elements are `test\_sanity.py` and `makecheck`:

```
wget http://boinc.berkeley.edu/source/nightly/boinc-cvs
-2004-09-08.tar.gz
tar fxvz boinc-cvs-2004-09-08.tar.gz
cd boinc
./configure && test/test_sanity.py
make
make check
```

Failure of `test\_sanity.py` is due to the authorizations of the MySQL server. The line "`make check`" implements a number of self-tests after "`make`". In the case of the source code of 8 September 2004, the first test was passed successfully, but the second test triggered the message "`Exception thrown - bug in test scripts?`". This in fact seems to be a case of a bug in the test script, because BOINC nevertheless turned out to work properly. Apart from this, the same problem also occurs in other up-to-date versions of the source code, which leads to the conclusion that the tests are no longer maintained.

6.3 Project setup

After successful compilation of the source code (in the home directory), creation of a scientific project in another directory (`/data/boinc`) can commence. I use the term "scientific project" to make a clear distinction between this project and the administrative project, which represents a part of my BOINC modification and the setup of which is described later on.

6.3.1 Final system preparation steps

The BOINC projects are to run under a separate UID, and therefore the system user 'boincsrv' is established:

```
adduser boincsrv
```

A MySQL user with the same name must be created:

```
mysql -u root -p
> GRANT Select, Insert, Update, Delete, Create, Drop ON
 *.* TO 'boincsrv'@'localhost'
```

A directory for the BOINC projects must be established, also properly configuring the corresponding rights:

```
mkdir /data/boinc
chgrp boincsrv /data/boinc/{projects,keys}
chmod g+w /data/boinc/{projects,keys}
```

6.3.2 The script "make_project"

The script "boinc/tools/make_project"serves the purpose of establishing projects. It creates a complete project directory comprising all files and sub-directories required, a complete project web and the database needed for the new project. A description of the parameters expected by the system can be obtained with "`make\_project --help`". This script is executed as user "boincsrv" (like all further commands needed to generate the project). In our case, a project called "hello" is created:

```
boincsrv@wtdist /home/kornberger/boinc/tools\$
./make_project --base /data/boinc --delete_prev_inst --
drop_db_first
--user_name boincsrv --url_base http://fsmtdist.ist.tu-
graz.ac.at/ hello
Creating project 'hello' (short name 'hello'):
           BASE = /data/boinc/
        KEY_DIR = /data/boinc/keys/
   PROJECT_ROOT = /data/boinc/projects/hello/
       URL_BASE = http://fsmtdist.ist.tu-graz.ac.at/
  HTML_USER_URL = http://fsmtdist.ist.tu-graz.ac.at/
  hello/
   HTML_OPS_URL = http://fsmtdist.ist.tu-graz.ac.at/
   hello_ops/
        CGI_URL = http://fsmtdist.ist.tu-graz.ac.at/
        hello_cgi/

Continue? [Y/n] y
Setting up server: creating directories

Keys don't exist in /data/boinc/keys/; generate them? [
Y/n] y
Setting up server files: linking cgi programs Done
installing default daemons.
Done installing files.

Steps to complete installation:

 Set permissions for Apache:

   cat /data/boinc/projects/hello/hello.httpd.conf >> /
   etc/apache/httpd.conf && apachectl restart
   # (path to httpd.conf varies)

 Add to crontab (as root)
   (If cron cannot run "start", try using a helper
   script to set PATH and
   PYTHONPATH)

    0,5,10,15,20,25,30,35,40,45,50,55 * * * * /data/
    boinc/projects/hello/bin/start --cron

To start, show status, stop BOINC daemons run:

    /data/boinc/projects/hello/bin/start
    /data/boinc/projects/hello/bin/status
    /data/boinc/projects/hello/bin/stop

Master         URL: http://fsmtdist.ist.tu-graz.ac.at/
hello/
Administration URL: http://fsmtdist.ist.tu-graz.ac.at/
hello_ops/

 Add Work Generator, Validator and Assimilator daemons
 for your applications


Further steps are necessary to add applications and
work.
See the online documentation at http://boinc.berkeley.
edu/
```

The script already specifies the next step:

- The Apache configuration must be adjusted (as user root):

```
cat /data/boinc/projects/hello/hello.httpd.conf >>
/etc/apache/httpd.conf && apachectl restart
```

- Then the UID is changed to "boincsrv", and `crontab -e` is executed to insert as follows in the crontab of the user:

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

  Now the cron jobs are executed under the UID "boincsrv".

6.3.3 The xadd step

The template "project.xml" is copied from the source tree into the project directory:

```
boincsrv@wtdist /home/kornberger/boinc\$ cp tools/
 project.xml /data/boinc/projects/hello/
```

Then it is edited. In this case, the supported platforms (Linux, Mac, Windows) are indicated. For the first test project, I specified only Linux support.

```
boincsrv@wtdist /data/boinc/projects/hello\$ vi project
.xml
<boinc>
    <platform>
        <name>i686-pc-linux-gnu</name>
        <user_friendly_name>Linux/x86</
        user_friendly_name>
    </platform>
    <app>
        <name>hello</name>
        <user_friendly_name>The famous hello world
        application</user_friendly_name>
    </app>
```

```
</boinc>
```

Then xadd is called in the project directory:

```
boincsrv@wtdist /data/boinc/projects/hello\$ bin/xadd
project.xml
Processing <Platform#None i686-pc-linux-gnu> ...
  Committed <Platform#1 i686-pc-linux-gnu> ; values:
{'_dirty': 0,
 '_lazy_lookups': {},
 'create_time': 1095775900L,
 'deprecated': 0L,
 'id': 1L,
 'name': 'i686-pc-linux-gnu',
 'user_friendly_name': 'Linux/x86'}
Processing <App#None hello> ...
  Committed <App#1 hello> ; values:
{'_dirty': 0,
 '_lazy_lookups': {},
 'create_time': 1095775900L,
 'deprecated': 0L,
 'id': 1L,
 'min_version': 0L,
 'name': 'hello',
 'user_friendly_name': ''}
```

6.3.4 Adjusting the file ”project.inc”

The file project.inc is edited in the project file:

```
boincsrv@wtdist /data/boinc/projects/hello\$
vi html/project/project.inc
define("PROJECT", "TheHelloWorldProject");
define("MASTER_URL", "http://fsmtdist.ist.tu-graz.ac.at
 /hello");
```

```
define("URL_BASE", "http://fsmtdist.ist.tu-graz.ac.at/
 hello");
define("IMAGE_PATH", "../user_profile/images/");
define("IMAGE_URL", "user_profile/images/");
define("PROFILE_PATH", "../user_profile/");
define("PROFILE_URL", "user_profile/");
define("LANGUAGE_FILE", "languages.txt");
define("STYLESHEET", "black.css");
define("COPYRIGHT_HOLDER", "The dIST-Team");
define("SYS_ADMIN_EMAIL", "bekor@gmx.net");
```

Then a minor optical feature in terms of the web interface is adjusted:

```
vi html/user/index.php
Line 53:  <link rel=stylesheet type=text/css href=black
.css>
Line 91:  <tr><td valign=top bgcolor=3232cc>
```

6.3.5 Securing the web

The web is secured by means of the two files `.htaccess` and `.htpasswd` in the project subdirectory `/data/boinc/projects/hello/html/ops`. `.htaccess` contains as follows:

```
[root@wtdist /data/boinc/projects/hello/html/ops]# cat
.htaccess
AuthName "BOINC ADMIN PAGE"
AuthType Basic
AuthUserFile /data/boinc/projects/hello/html/ops/.
htpasswd
Require valid-user
```

The following command in the directory to be secured serves to generate the password file `.htpasswd` with the user name ”admin”:

```
htpasswd -c .htpasswd admin
```

The selected password is entered twice, thus securing the web. The visible success of the steps executed so far consists in that the two pages

- `http://fsmtdist.ist.tu-graz.ac.at/hello` and
- `http://fsmtdist.ist.tu-graz.ac.at/hello_ops`

are accessible via the browser, the latter page being password-protected.

6.3.6 Starting the project

The project can be started in the project directory using:

```
boincsrv@wtdist:~/projects/hello\$ ./bin/start

Entering ENABLED mode

Starting daemons
  Starting daemon: feeder -d 3
  Starting daemon: transitioner -d 3
  Starting daemon: file_deleter -d 3
```

This shows the start up procedure of the project "hello", where the `feeder`, the `transitioner` and the `file_deleter` program are started with the option "-d 3" meaning debug level 3. This debug level produces a large output and should be used only during the initial phase of a project. See Section 6.3.8 for details.

6.3.7 Debugging

In the event that the project fails to work properly, a few methods can be recommended to track down the error:

- `./bin/status` can be entered to check whether or not the project is running.
- The sub-directory `./log\_<host>/` additionally contains the logfiles of the started programs.

- My first attempt to set up a server on Mandrake_linux (Mandrake 10.0) failed because the feeder was not running. According to information provided by a BOINC developer in response to a corresponding inquiry, this failure was most probably due to a shared memory problem. However, my configuration efforts in this respect were unsuccessful. The problem does not occur when Debian Woody with kernel 2.4.20-bf2.4 or kernel 2.6.7 is used.

- Database entries can be visualized very efficiently using the software phpmyadmin. This software can be configured in such a way that the BLOB fields of the databases where the projects store their XML files are also shown. Additionally, the databases can be administrated via the BOINC web interface by calling the URL `http://fsmtdist.ist.tu-graz.ac.at/hello_ops/` with the web browser.

- By uncommenting the symbol `SHOW_QUERIES` in `db/db\base.C`, and recompilation, all database queries are written to stderr (for daemons, this goes to log files; for command-line apps it is written to the terminal). This is verbose but extremely useful for tracking down database-level problems.

6.3.8 Reducing the log level

Each project directory contains a file `config.xml`:

```
[...]
<upload_url>
      http://wtdist/hello_cgi/file_upload_handler
    </upload_url>
  </config>
  <tasks/>
  <daemons>
    <daemon>
      <cmd>
        feeder -d 3
      </cmd>
    </daemon>
    <daemon>
      <cmd>
        transitioner -d 3
      </cmd>
    </daemon>
    <daemon>
      <cmd>
        file_deleter -d 3
      </cmd>
    </daemon>
  </daemons>
</boinc>
```

Starting with log level 3, which is very favourable for initial debugging, is specified for the three programs `feeder`, `transitioner` and `file_deleter`. For practical operation of projects, however, this log level is practically unsuitable as logfiles comprising several gigabytes are created within a few weeks. For information about where these logfiles can be found see Section 4.1. The debug levels are: 1 = critical messages only, 2 = normal messages, 3 = detailed debugging info.

6.4 Scientific application

6.4.1 Source code

As the "scientific" application for the first test, I have chosen a distributed HelloWorld program. The source code for this program can be downloaded from [21]. An error in this example that was difficult to identify resulted from the fact that the major version number of the application was 3. This caused a problem due to a non-documented or at least insufficiently documented relation included in BOINC concerning the file names of the scientific applications:

1. The file name of each scientific application must contain its version number. Further rules regarding the allocation of names can be found in the BOINC

documentation.

2. The major version number part of the version number must correspond to that of the core client. This means that a core client with a version number 4.05 would receive workunits that can be computed with hello_4.08 but would not receive workunits if the name of the scientific application were hello_3.08.

Attention: If this error occurs, the answer from the server is: "`No work available. There was work for other platforms`" and no further indication is given that the problem has resulted from the version number. Consequently, the cause for this error was extremely hard to find and I had to analyze the source code to identify it. But as this is rather a feature than a bug, I didn't change the behavior. It suffices if one knows this special property and how to interpret the message from the server.

6.4.2 Compiling the scientific application

The source code of the scientific application is unpacked. In the makefile, the desired version number is defined and the paths to the libraries required are adjusted if necessary. Then `make` is called.

```
[root@wtdist /home/kornberger/hello.d]# make
g++  -I../../lib  -I../../api   -include ../../config.h
      -c  hello.C -o hello.o
g++ -L../../lib -lboinc  -o hello   -static  hello.o
  ../../api/boinc_api.o -L../../lib -lboinc
strip hello
ln -f hello hello_4.08_i686-pc-linux-gnu
```

In the context of one's own applications, it is essential to remember that the scientific application must always be statically linked so that it can run on all systems. The libraries I needed for static linking under Debian seem to be part of the standard scope of installation and already existed on the system. Under Mandrake, at least the following libraries had to be installed: `urpmi libstdc ++5-static-devel glibc-static-devel`.

6.4.3 Registering the application with the project

The scientific application and the core client are copied into the project directory or, more precisely, a sub-directory of `/apps` that must have the same name as the application itself:

```
mkdir /data/boinc/projects/hello/apps/hello
mkdir /data/boinc/projects/hello/apps/boinc
cp hello_4.00_i686-pc-linux-gnu /data/boinc/projects/
hello/apps/hello/
cp ../../client/boinc_4.50_i686-pc-linux-gnu /data/
boinc/projects/hello/apps/boinc/
```

Then "`./bin/update_versions`" is called and it is essential that this step is performed from the project's root directory. Otherwise, the paths are not resolved properly. Prior to this, the project must be stopped using `./bin/stop` if it is active. If this step is repeated later again because a new application is added to the project, the lines adding the Boinc client need not be executed anymore.

```
[root@wtdist /data/boinc/projects/hello]# bin/
update_versions
Looking for core versions in /data/boinc/projects/hello
/apps/boinc
Found core version 450 for <Platform#1 i686-pc-linux-
gnu>: boinc_4.50_i686-pc-linux-gnu
Copying boinc_4.50_i686-pc-linux-gnu to /data/boinc/
projects/hello/download/boinc_4.50_i686-pc-linux-gnu
Looking for <App#1 hello> versions in /data/boinc/
projects/hello/apps/hello
Found <App#1 hello> version 400 for <Platform#1 i686-pc
-linux-gnu>: hello_4.00_i686-pc-linux-gnu

SECURITY WARNING:
=================
You have not provided a signature file for /data/boinc/
projects/hello/apps/hello/hello_4.00_i686-pc-linux-gnu.

I can generate one now, but this is highly
unrecommended.
Generating code signatures on network-connected
computers is
a security vulnerability, and should not be done for
publicly-accessable projects.

Continue with automatically generating a code signature
?
[y/N] y
Signing /data/boinc/projects/hello/apps/hello/hello_4
.00_i686-pc-linux-gnu
Copying hello_4.00_i686-pc-linux-gnu to /data/boinc/
projects/hello/download/hello_4.00_i686-pc-linux-gnu
Ready to commit 2 items:
    <CoreVersion#None 450 i686-pc-linux-gnu>
    <AppVersion#None hello 400 i686-pc-linux-gnu>
Continue [Y/n] y
Committed:
    <CoreVersion#1 450 i686-pc-linux-gnu>
    <AppVersion#1 hello 400 i686-pc-linux-gnu>
Done
```

Attention: BOINC uses digital signatures to allow the core client to authenticate executable files. I ignored the security warning above for our first HelloWorld project, but for real publicly available projects this issue must be taken into account. Otherwise, if hackers break into the BOINC server, they are able to use BOINC to distribute malicious code to the participants' machines. BOINC's documentation [5] recomments the following procedure:

- One computer should be used as code signing machine. The program `boinc`

`/lib/crypt_prog` must be installed on this machine. This computer must remain physically secure and disconnected from the network. A CD-RW or USB stick can be used for moving files to and from this computer.

- `crypt_prog -genkey` is used to create a code-signing key pair. The public key must be copied to the BOINC server. The private key is to be kept on the code-signing machine. A copy of the key pair should be made (e.g. on a CD-ROM that must be kept locked up), and all other copies of the private key must be deleted.
- An executable file is signed by moving it to the code signing machine, running `crypt_prog -sign` to produce the signature file and then moving the signature file to the server. Then `update_versions` is used on the server to install the application, including its signature files, in the download directory and database.

6.5 Generating a workunit (WU)

The directory hello.d of the HelloWorld application contains, among other things, the files `hello_re.xml` and `hello_wu.xml`. These are copied directly into the project directory, i.e., in our case into `/data/projects/hello`, where they are modified as follows:

```
boincsrv@wtdist:~/projects/hello\$ vi hello_re.xml
<!-- result template for the BOINC hello world program
-->
<file_info>
    <name><OUTFILE_0/></name>
    <generated_locally/>
    <upload_when_present/>
    <url>http://fsmtdist.ist.tu-graz.ac.at/hello_cgi/
    file_upload_handler</url>
    <max_nbytes>100000</max_nbytes>
</file_info>
```

```
<result>
    <file_ref>
        <file_name><OUTFILE_0/></file_name>
        <open_name>out</open_name>
    </file_ref>
</result>
```

```
boincsrv@wtdist:~/projects/hello\$ vi hello_wu.xml
<!-- workunit template for the BOINC hello world
program -->
<workunit>
  <min_quorum>           1  </min_quorum>
  <target_nresults>      1  </target_nresults>
  <max_error_results>    99  </max_error_results>
  <max_total_results>   10  </max_total_results>
  <max_success_results>  5  </max_success_results>
  <rsc_fpops_est>      2e9  </rsc_fpops_est>
  <rsc_fpops_bound>    9e10  </rsc_fpops_bound>
  <rsc_memory_bound> 100000000 </rsc_mem_bound>
  <rsc_disk_bound>   100000000 </rsc_disk_bound>
  <delay_bound>        3600 </delay_bound>
</workunit>
```

The meaning of the parameters indicated above is explained in Section 4.3. An important precondition, which is unfortunately not expressly indicated in the BOINC documentation, is that the upload URL must point to the `file_upload_handler` as shown in the file `hello_re.xml` above. This simple HelloWorld workunit does not use an input file. In the event that input files are needed, this is also specified there, and the input files are put into the directory `/download`. The utility `./bin/create_work` is used to enter this workunit into the database.

```
boincsrv@wtdist:~/projects/hello\$ ./bin/create_work -
appname hello -wu_name firstHelloWU -wu_template
hello_wu.xml -result_template hello_re.xml;done
```

Alternatively, workunits can be inserted by means of a separate program, two libraries containing the functions required for this purpose being available:

- crypt.C provides a function for reading the file upload authentication key:

```
int read_key_file(char* path, R_RSA_PRIVATE_KEY&
key);
```

- backend_lib.C,h provides the function create_work(), which creates a workunit and one or several results. The arguments are similar to those of the utility program, but some of the information is passed in the WORKUNIT structure, namely the fields name, appid, batch, rsc_fpops, rsc_iops, rsc_memory, rsc_disk and delay_bound.

```
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* infile_dir,                     //
    where input files are
    const char** infiles,                       //
    array of input file names
    int ninfiles
    R_RSA_PRIVATE_KEY& key,               // upload
    authentication key
    SCHED_CONFIG&
);
```

6.6 In operation

6.6.1 The client

Participation is very simple:

- Visit the website of the project, e.g. `http://fsmtdist.ist.tu-graz.ac.at/hello`
- Generate an account. The project automatically sends an e-mail containing a 32 bit key.
- Enter the 32 bit key in the web interface to confirm the account.
- Enter the desired settings in the corresponding form.
- Download the client from the website of the project, move it into a separate directory and make it operable using `chmod a+x boinc\_client`.
- Start the client and indicate the master URL as well as the 32 bit key. See Section 6.7.1 for simplifications implemented in the modified core client.

After indication of the project URL and the ID, the client, during the first session, indicates that it is performing preparatory activities such as benchmarks, etc. Then it contacts the scheduler by transmitting a query to `http://fsmtdist.ist.tu-graz.ac.at/hello_cgi/cgi`. The entry `ScriptAlias /hello_cgi /data/boinc/projects/hello/cgi-bin` in the Apache configuration file `etc/apache/httpd.conf` causes the query to be addressed to the program `cgi` in the project directory. The scheduler provides a reply, and the client receives the general settings as well as the application `hello_4.00_i686-pc-linux-gnu`. Subsequently, the workunit is computed and uploaded back to the server.

```
[root@polar client_boinc]# ./boinc_4.50_i686-pc-linux-
gnu
2004-09-22 10:40:45 [---] Starting BOINC client version
 4.50 for i686-pc-linux-gnu
2004-09-22 10:40:45 [hello] Project prefs: using your
defaults
```

```
2004-09-22 10:40:45 [hello] Host ID not assigned yet
2004-09-22 10:40:45 [---] No general preferences found
- using BOINC defaults
2004-09-22 10:40:45 [---] Running CPU benchmarks
2004-09-22 10:40:45 [---] Suspending computation and
network activity - running CPU benchmarks
2004-09-22 10:41:46 [---] Benchmark results:
2004-09-22 10:41:46 [---]    Number of CPUs: 1
2004-09-22 10:41:46 [---]    508 double precision MIPS
(Whetstone) per CPU
2004-09-22 10:41:46 [---]    1234 integer MIPS (
Dhrystone) per CPU
2004-09-22 10:41:46 [---] Finished CPU benchmarks
2004-09-22 10:41:47 [---] Resuming computation and
network activity
2004-09-22 10:41:47 [---] Insufficient work; requesting
 more
2004-09-22 10:41:47 [---] Insufficient work; requesting
 more
2004-09-22 10:41:47 [hello] Requesting 17280 seconds of
 work
2004-09-22 10:41:47 [hello] Sending request to
scheduler: http://fsmtdist.ist.tu-graz.ac.at/hello_cgi/
cgi
2004-09-22 10:41:47 [hello] Scheduler RPC to http://
fsmtdist.ist.tu-graz.ac.at/hello_cgi/cgi succeeded
2004-09-22 10:41:47 [hello] General preferences have
been updated
2004-09-22 10:41:47 [---] General prefs: from hello (
last modified 2004-09-22 09:53:02)
2004-09-22 10:41:47 [---] General prefs: no separate
prefs for home; using your defaults
```

```
2004-09-22 10:41:47 [hello] Project prefs: no separate
prefs for home; using your defaults
2004-09-22 10:41:47 [hello] Started download of hello_4
.00_i686-pc-linux-gnu
2004-09-22 10:41:47 [---] May run out of work in 0.10
days; requesting more
2004-09-22 10:41:47 [hello] Requesting 17276 seconds of
 work
2004-09-22 10:41:47 [hello] Sending request to
scheduler: http://fsmtdist.ist.tu-graz.ac.at/hello_cgi/
cgi
2004-09-22 10:41:48 [hello] Scheduler RPC to http://
fsmtdist.ist.tu-graz.ac.at/hello_cgi/cgi succeeded
2004-09-22 10:41:48 [hello] Message from server: No
work available
2004-09-22 10:41:48 [hello] No work from project
2004-09-22 10:41:48 [hello] No work from project
2004-09-22 10:41:48 [hello] Deferring communication
with project for 1 hours, 0 minutes, and 0 seconds
2004-09-22 10:41:48 [hello] Deferring communication
with project for 1 hours, 0 minutes, and 0 seconds
2004-09-22 10:41:52 [hello] Finished download of
hello_4.00_i686-pc-linux-gnu
2004-09-22 10:41:52 [hello] Throughput 105567 bytes/sec
2004-09-22 10:41:52 [hello] Starting result
secondHelloWU_0 using hello version 4.00
2004-09-22 10:42:10 [hello] Computation for result
secondHelloWU finished
2004-09-22 10:42:10 [hello] Started upload of
secondHelloWU_0_0
2004-09-22 10:42:10 [hello] Finished upload of
secondHelloWU_0_0
2004-09-22 10:42:10 [hello] Throughput 1918 bytes/sec
```

Then the result is visible in the upload directory on the server side:

```
boincsrv@wtdist:~/projects/hello/upload\$ cat
secondHelloWU_0_0
Hello, BOINC World!
Stress test begins...
Stress test ends...
```

6.7 Operation of the administrative project

The previous paragraphs of the present section section described setup and operation of a normal BOINC server. As soon as this server is running smoothly, the modifications I have programmed for the dIST project can be integrated. For this purpose, it is essential to distinguish between the code for the client and the code for the server.

6.7.1 The modified client

The modifications performed in the core client are based on the source code `boinc-cvs-2004-09-08.tar.gz` and are largely implemented in the two newly added files `austroBoinc_utils.h` and `austroBoinc_utils.C`. Some #includes and #defines however are also concerning other sections of the code. In the framework of my diploma thesis, I make both the original code and the modified code available on a CDR. The modifications contained therein exclusively serve the purpose of compiling a modified core client. Before starting `./configure && make`, two adjustments must be performed in the source code if required.

1. As the modified client is designed for use with the administrative project, it would not make sense to ask the user for the master URL as in the original core client. On the other hand, the modified core client, being used as a normal application by the dIST project, may only consist of one single executable file. Therefore, I have decided to compile the master URL of the administrative project as a fixed component into the core client, which is

the only possible rather than an elegant solution. Prior to compiling, the source code must be changed in this respect where a client operated with an administrative project on a different BOINC server is to be established. The respective section of the code can be found in the file `boinc/client/austroBoinc_utils.h`:

```
// This is the url of the AustroBoinc-enhanced
project which
// only does administrative tasks
#define ADMINISTRATIVE_PROJECT "dist"
#define ADMINISTRATIVE_URL "fsmtdist.ist.tu-graz.ac
.at"
#define ADMINISTRATIVE_PROJECT_STRING "http://
fsmtdist.ist.tu-graz.ac.at/dist/"
```

2. The same file contains an entry defining how often the core client is to inquire with the server whether or not there are new projects to support or whether or not the `resource_share` rate must be changed. The value entered indicates the number of seconds that must expire before the next query is transmitted. For test purposes, this value can be set to 10 seconds. In a practical operation setting, a value of 24 hours can be recommended.

```
// Wait at least this time to request informations
about new projects.
#define TIME_NEW_PROJECT_REQUEST 10
```

After performing these adjustments, the command `make`, which creates the new core client in the directory `boinc/client`, can be executed in the directory `boinc`.

6.7.2 The modified server

Only a few minor changes are required to create an administrative project instead of a normal project.

1. A normal project with the name ”dist” is generated as described in Section 6.3. Experience has shown that a significant error source in this respect can be eliminated by refraining from using capital letters in the project name as the project name entered will occur frequently in different contexts in the course of the further setup procedure.
2. Subsequently, the following files are copied from the PHP directory on the CDR to `dist/html/user/` in the project directory.
   - `automatic_subscription.php`
   - `automatic_account_generation.php`
3. The file `automatic_account_generation.php` must be adjusted in two respects:

```
<?php
#  automatic_account_generation.php
#
#  Mon Nov  1 21:41:34 2004
#  Copyright  2004  Bernhard Kornberger
#  Email: bekor@gmx.net
#
#
#  This program is distributed in the hope that it
will be useful,
#  but WITHOUT ANY WARRANTY; without even the
implied warranty of
#  MERCHANTABILITY or FITNESS FOR A PARTICULAR
PURPOSE.

#  Purpose of this file: This is a feature for
simplified subscription.
#  One doesn't have to subscribe via the web
interface anymore. Instead
```

```
#  the user data can be specified in the command
line:
#  -autoFirstTime <username> <mail> <country> <
postal code> <secret>
#  The modified client (we call it the AustroBoinc-
Version) sends the
#  parameters to this script which generates a new
user account.
#
#  Of course this opens the door for DOS attacks
because someone could
#  auto-generate thousands of user accounts. Two
measures are destined
#  to prevent such misuse:
#
#  1) Disable this php script after the initial
installation phase.
#  2) To use this feature one has to know the
shared secret key (tan -
#  transaction number) below. Give it only to
trusted persons and
#  change it frequently.
#
#  To clarify the leading thought behind the
AustroBoinc-Mod: This is
#  not intended for use with big projects like Seti
 but rather for cases
#  where a few hundred computers of a university
are to be managed.

$tan="ChangeThis";
$admin_url="fsmtdist.ist.tu-graz.ac.at/dist";
```

Here, the line $tan=... must be adjusted, the string in quotation marks representing the password that must be entered for simplified registration as described in Section 5.3. If any other administrative project than that under the master URL `fsmtdist.ist.tu-graz.ac.at/dist` is used, this line must be adjusted as well. To ensure correct functioning of this script, all projects to be supported must be on the same server as the administrative project dIST. This is due to the fact that this script manipulates the databases of the scientific projects and is only designed for local access.

Chapter 7

Real applications

The previous chapters of this diploma thesis describe the dIST system. This chapter, in contrast, will provide an impression of the different kinds of problems that can be solved at the IST [17] by means of the dIST system. The basis of most of the calculations performed at the IST is a database containing order types. Therefore, the concept of order types will be explained before dealing with actual applications.

7.1 Order types

Let $S = \{p_1, ..., p_n\}$ be a set of n points in the plane. In this context, one might want to know how many different ways to arrange the n points exist, and the answer to this question is of course: Infinitely many. But for a large number of problems it is absolutely sufficient to consider just the combinatorial properties of a point set rather than its metric properties. These are given by the crossing properties of the complete graph of a point set as shown in figure 7.1 where one can see that for instance there are just two possibilities of arranging four points in the plane so as to achieve different crossing properties whereas there are three options for five points. The order type of each point set $S = \{p_1, ..., p_n\}$ is determined by a mapping that assigns to each index triple $\{i, j, k\}$ the orientation (clockwise or counter-clockwise) of the points p_i, p_j, p_k. For this purpose it is required that no three points lie on a straight line. It is easy to see that the crossing properties

n	3	4	5	6	7	8	9	10	11
Planar Order Types	1	2	3	16	135	3 315	158 817	14 309 547	2 334 512 907

Table 7.1: Numbers of different order types of size n

of two point sets are equivalent only if the point sets have the same order type and that a change in the orientation of three arbitrary points $\{p_i, p_j, p_k\}$ would lead to different crossing properties. At the IST a considerable amount of work has been done by Oswin Aichholzer and Hannes Krasser to establish a complete database of such order types for sets with up to 11 points, as summarized in [3] where it turned out that the number of different arrangements of n points in the plane increases dramatically with increasing n as shown in table 7.1. In [18] order types are described in a more scientific way and in far more detail, but for the purpose of this thesis the description given above should be sufficient.

7.2 Happy End Problem

A very famous problem, mentioned also in the literature [14], is the Happy End Problem. In 1935 Paul Erdös and György Szekeres set themselves the task of determining the smallest positive number $g(n)$, such that any set S of at least $g(n)$ points, $S = \{p_1, ..., p_{g(n)}\}$, in general position in the plane contains n points that are the vertices of a convex n-gon. They also conjectured that $g(n) = 2^{(n-2)} + 1$ for all $n \geq 3$. Szekeres proved an upper bound and in recent years this upper bound has been improved so that we know that $g(n) \leq \binom{2n-5}{n-2} + 2$. However, 70 years after the problem was stated it is still unresolved for $n > 5$, where the conjecture is that $g(6) = 17$ and the best known upper bound is $g(6) \leq 37$.

To answer this conjecture there was an attempt by Birgit Breitenlechner [4] to approach the Happy End Problem for $n = 6$ by computing whether or not there is an order type of cardinality 17 with no convex hexagon. As shown in table 7.1, the number of order types grows exponentially with increasing n and it is not even feasible to build a complete database of order types of cardinality 12.

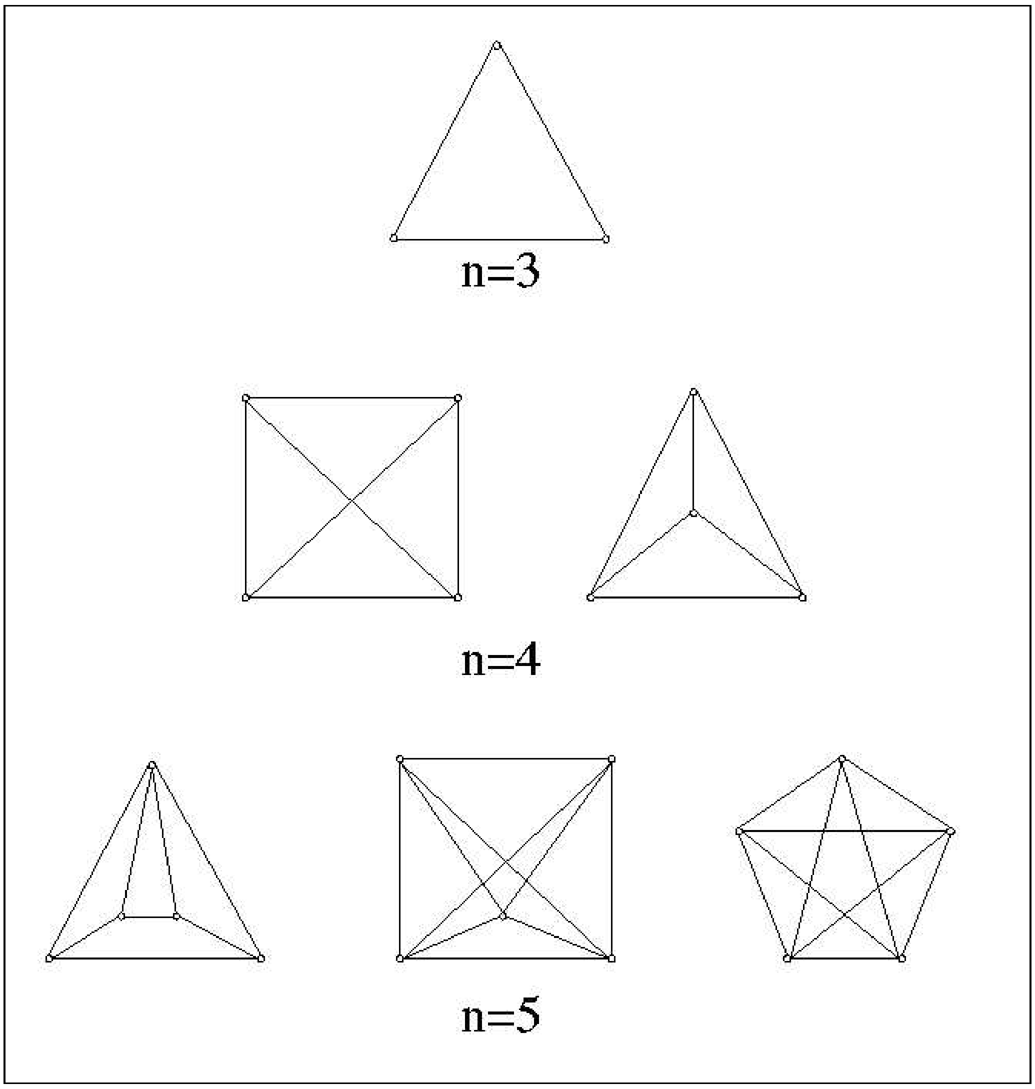

Figure 7.1: Different order types of point sets with cardinality n

Therefore the order types of cardinality 17 are derived by applying an abstract extension technique to the order types from the database described in section 7.1. It is fortunately not necessary to extend all the 2 334 512 907 order types of cardinality 11 but only the ones containing no convex hexagon, which reduces the number of point sets to be considered to 235 987 328. This extension leads to order types of cardinality 12 whereof in turn only those containing no convex hexagon need to be extended to 13 points and so on. Because of the exponential growth of existing order types, this attempt was based on the hope that the number of order types without convex hexagons will decrease with the number of points as least as significantly as the number of overall order types increases. Unfortunately, this turned out to be an illusion as for instance the extension of one specific 11-point order type generated more than 142 millions of 16-point order types without convex hexagons and this example seems not to be an exception. A CPU with 1 GHz needed more than two weeks to extend this specific point set to cardinality 16 and considering such computation times we estimate that the task of computing all order types of cardinality 17 could take as much as 100 years yet on a modern personal computer.

This is where dIST comes into play. The task of computing each order type of cardinality 17 is perfectly suited to be divided into small sub-tasks which can be solved by the participants of our new distributed computing system. We will slightly modify the algorithms implemented by Birgit Breitenlechner to make them suitable for dIST. With participation of all computers of the computing rooms of the TUG and contribution of computing power from students, we could achieve to complete this task within several months.

7.3 Counting triangulations

Counting the number of triangulations of a point set in the plane represents another task requiring an enormous amount of computing capacity. Exact numbers are known, and can be retrieved from the database, for all sets comprising $n \leq 11$ points. The currently best general asymptotic lower bound for this problem can be derived from these results for small sets, see [1]). This bound can be improved by obtaining a tight lower bound for order types of higher cardinality, e.g. $n = 12, 13, ...$ By adding an interior point to a given set, the number of triangulations increases by a constant factor of at least $c = 2$, which means that the starting point for determining the tight lower bound for $n = 12, 13, ...$ is the 2 351 11-point sets each achieving a maximum of 1 118 triangulations. This seems to be feasible even for $n = 13$, as only 845 829 sets with a maximum of 1 786 triangulations need to be taken into account, which represents less than 0.037% of the entire database of 11 points. With our new distributed computing network, additional results will be available soon.

7.4 Decomposition

Another task is to search for optimal decompositions of small point sets. In this context it was asked for disjoint empty convex polygons spanned by the set, see [2] for details. Two results of these investigations were:

- Any set of 8 points contains either an empty convex pentagon or two independent empty convex quadrilaterals.
- Any set of 11 points contains either an empty convex hexagon or an independent empty convex quadrilateral.

These results directly lead to an upper bound of $7n/10$ for the number of convex or pseudotriangular faces used to decompose a set of n points. There is hope to extend these results for sets of bigger size using the dIST system and applying an extension technique to the order types of our data base.

7.5 Rectilinear crossing numbers

In order to get crossing numbers for point sets with $n > 11$ points it is not sufficient to extend only the order types with optimal drawings, as not all optimal drawings of K_n (i.e., a complete graph of order n) contain an optimal sub-drawing of K_{n-1}. In fact, it is not even known whether there always exists at least one optimal drawing of K_n which contains an optimal sub-drawing of K_{n-1}. For an arbitrary rectilinear drawing of K_n each of its n sub-drawings K_{n-1} has at least

number of crossings	102	104	106	108	110	112
number of order types	374	3 984	17 896	47 471	102 925	228 497

Table 7.2: Number of order types of K_{11} having a specific small crossing number

$\overline{cr}(K_{n-1})$ crossings. Summing up the crossing numbers in the n sub-drawings K_{n-1} each crossing is counted n-4 times, as each quadruple of points determining a crossing shows up in all but 4 of the sub-drawings K_{n-1}. This implies the relation

$$\overline{cr}(K_n) \geq \left\lceil \frac{n}{n-4} \overline{cr}(K_{n-1}) \right\rceil$$

As a consequence, for any drawing of K_n with c crossings there exists at least one sub-drawing K_{n-1} with a maximum of $\lfloor \frac{n-4}{n} c \rfloor$ crossings. For example, a drawing of K_{12} with 153 or less crossings has to contain at least one sub-drawing of size 11 with $\lfloor \frac{12-4}{12} 153 \rfloor = 102$ crossings.

There is also a parity property that leads to further improvements for sets of odd cardinality: Let $n \in \mathbb{N}$ be odd and consider a straight-line drawing of K_n with c crossings. Then

$$c \equiv \binom{n}{4} (mod 2)$$

A drawing of K_{13} with 229 (or less crossings) contains at least one sub-drawing K_{12} with $\lfloor \frac{9}{13} \rfloor = 158$ (or less crossings) and recursive application shows that there exists a sub-drawing of size 11 with $\lfloor \frac{8}{12} 158 \rfloor = 105$ crossings. By the parity property we can further reduce the number of crossings for the 11-point subset to a maximum of 104. Thus, to achieve a database of all order types of cardinality 13 with 229 (or fewer) crossings, we can take the order types from our complete database of order types of size 11 with a maximum of 104 crossings, i.e., either 102 or 104 crossings. Table 7.2 shows the number of order types of K_{11} having a certain low number of crossings. The conclusion is that for this example we just need to consider 374 + 3 984 order types instead of all 2 334 512 907 order types, i.e., less than 0.0002 %.

Using the above mentioned properties the rectilinear crossing numbers for $n = 12, ..., 17$ have already been computed (see table 7.3). To provide an impression of the duration of the computations: The computations to determine $\overline{cr}(K_n)$ and

n	12	13	14	15	16	17
$\overline{cr}(K_n)$	153	229	324	447	603	798
d_n	1	4 534	20	16 001	36	≥37 269

Table 7.3: Rectilinear crossing number $\overline{cr}(K_n)$ for $n = 12, ..., 17$ and the number d_n of order types with $\overline{cr}(K_n)$

d_n for $n \leq 16$ have taken 58 days on a 2 GHz CPU. The average computing time for sets of size 11 was approximately 3.7 hours, the slowest 11-set needed 56 hours and the fastest 11-sets took only 16 seconds. This variation in context with distributed computing is an additional problem because:

- Checkpointing, i.e. saving of intermediate results for later continuation, is very hard to implement when abstract order type extension is applied.
- It is useless to send results (remember, results are instances of computations, regardless of their computation status) with a huge demand of computation time to computers turned off every few hours.
- We don not know a result's computing time in advance.

At the same time I am writing this diploma thesis, Thomas Uttenthaler checks out our new dIST system within the scope of a project work at the IST [17]. For this purpose he applies the *cape* algorithm implemented by Birgit Breitenlechner, see [4], to order types of cardinality 11 with 104 crossings. He generated 445 workunits, each representing one order type (the outcome of the rest is already known from previous computations). His workaround regarding the problem described above is to perform three runs extending the sets up to $n = 20$:

- First run: The value `rsc_fpops_bound` (see Section 4.3) is set in such a way that results requiring more than approximately 5 hours (on a Pentium 4 with 3 GHz) are aborted. This run should finalize a big number of fast results so that the corresponding workunits will not have to be considered anymore. However, our intermediate results show that this bound causes 70 percent of the results to terminate before the computations are finished. In this turn the results are sent to arbitrary machines.

- Second run: The remaining workunits are regenerated, where the value `rsc_fpops_bound` is set in such a way that results requiring more than 24 hours are aborted. The hope is that this turn will leave just a small number of unsolved results.

- Third run: If there are still unsolved workunits after the first two runs, new results are generated for them, whereby `rsc_fpops_bound` is set to such a high value, that computations may take 500 hours. These results may only be sent to fast computers running 24/7, i.e. 24 hours a day and 7 days a week, which are the computers of the IST.

Chapter 8

Conclusion

At the beginning of this diploma thesis, my target was to develop a system for distributed computing. In particular, this system was planned to be used to solve the Happy End Problem, described in Section 7.2. I have evaluated existing systems for similar tasks and came upon BOINC, which is released under the Open Source license LGPL. I programmed a number of new functions extending BOINC so as to allow projects to dynamically evolve and terminate without the need to reconfigure the computers of the participants. In other words, the computers of the participants support not only one particular project but every project we specify. With the support and the know-how of my colleagues Bernd Haug and Thomas Uttenthaler, I was finally able to solve all problems, and now there is a fully functional, general purpose, distributed computing system installed at the IST, including all customized extensions required by our specific tasks. While I am finalizing the present thesis, several computers are busy calculating the crossing number problem described in Section 7.5.

Bibliography

[1] O. Aichholzer, F. Hurtado, and M. Noy, *A lower bound on the number of triangulations of planar point sets*, Computational Geometry: Theory and Applications, 29(2):135-145, 2004

[2] O. Aichholzer, C. Huemer, S. Renkl, B. Speckmann, and C. D. Toth, *On Pseudo-Convex Decompositions, Partitions, and Coverings*, 21st European Workshop on Computational Geometry (EWCG), Eindhoven, The Netherlands, pp. 1-2, 2005.

[3] O. Aichholzer, F. Aurenhammer, and H. Krasser. *Points and Combinatorics* Journal Telematik, 1:12-17, 2002.

[4] Birgit Breitenlechner, *Abstract Order Type Extension for Combinatorial Problems*, Master Thesis, Graz University of Technology, 2004.

[5] BOINC - Homepage and Documentation:
`http://boinc.berkeley.edu/`

[6] David P. Anderson, *Public Computing: Reconnecting People To Science*, Space Sciences Laboratory, University of California - Berkeley, March 21, 2004, `http://boinc.berkeley.edu/boinc2.pdf`

[7] David P. Anderson, *BOINC: A System for Public-Resource Computing and Storage*, Space Sciences Laboratory, University of California Berkeley, 2004, `http://boinc.berkeley.edu/grid_paper_04.pdf`

[8] BOINC Homepage: `http://boinc.berkeley.edu`

[9] ClimatePrediction.net: `http://climateprediction.net/`

[10] OMG CORBA Specification:
`http://www.omg.org/technology/documents/formal/corba_iiop.htm`

[11] Homepage of dIST: `http://fsmtdist.ist.tu-graz.ac.at/dist`

[12] Einstein@home: `http://boinc.de/einstein.htm`

[13] GNU General Public License, Free Software Foundation (FSF) 1991, `http://www.gnu.org/copyleft/gpl.html`

[14] Paul Hoffman, *The Man Who Loved Only Numbers*, Econ Ullstein List Verlag, München, 2001, ISBN 3-548-75058-3.

[15] Integrade: `http://gsd.ime.usp.br/integrade/`

[16] University of Sao Paulo: Andrei Goldleger, Fabio Kon, Alfredo Goldman, Marcelo Finger, Germano Capistrano Bezerra, *Integrade: object-oriented Grid middleware leveraging idle computing power of desktop machines*, `http://gsd.ime.usp.br/publications/cpe03_integrade.pdf`

[17] Institute for Softwaretechnology, Graz `http://www.ist.tugraz.at/`

[18] Hannes Krasser, *Order Types of Point Sets in the Plane*, PhD Thesis, Institute for Theoretical Computer Science, Graz University of Technology, Austria, October 2003.

[19] GNU Lesser General Public License, Free Software Foundation (FSF) 1991, `http://www.gnu.org/copyleft/lesser.html`

[20] LHC@home: `http://boinc.de/lhcathome.htm`

[21] Eric Myers' BOINC downloads:
`http://noether.vassar.edu/pub/myers/src/boinc/`

[22] OpenMosix: `http://openmosix.sourceforge.net/`

[23] Predictor@home: `http://boinc.de/predictor.htm`

[24] Pirates@home:
`http://pirates.vassar.edu/help/boinc-on-linux.html`

[25] Radio telescope Arecibo:
`http://www.naic.edu/public/the_telescope.htm`

[26] QADPZ: `http://qadpz.sourceforge.net/`

[27] SETI@home: `http://setiathome.ssl.berkeley.edu/`

[28] University of California, Berkeley: `http://www.berkeley.edu/`