SZTAKI Desktop Grid: a Modular and Scalable Way of Building Large Computing Grids

Zoltán Balaton[1], Gábor Gombás[1], Péter Kacsuk[1], Ádám Kornafeld[1], József Kovács[1], Attila Csaba Marosi[1],Gábor Vida[1], Norbert Podhorszki[2], Tamás Kiss[3]

[1]MTA SZTAKI Computer and Automation Research Institute of the Hungarian Academy of Sciences
H-1528 Budapest, P.O.Box 63, Hungary
{balaton,gombasg,kacsuk,kadam, smith,atisu,vida}@sztaki.hu

[2]University of California, Davis
Computer Science Department
2063 Kemper Hall, 1 Shields Avenue,
Davis CA 95616, USA
pnorbert@cs.ucdavis.edu

[3]University of Westminster
Cavendish School of Computer Science
115 New Cavendish Street, London W1W 6UW, UK
T.Kiss@westminster.ac.uk

Abstract

So far BOINC based desktop Grid systems have been applied at the global computing level. This paper describes an extended version of BOINC called SZTAKI Desktop Grid (SZDG) that aims at using Desktop Grids (DGs) at local (enterprise/institution) level. The novelty of SZDG is that it enables the hierarchical organisation of local DGs, i.e., clients of a DG can be DGs at a lower level that can take work units from their higher level DG server. More than that, even clusters can be connected at the client level and hence work units can contain complete MPI programs to be run on the client clusters. In order to easily create Master/Worker type DG applications a new API, called as the DC-API has been developed. SZDG and DC-API has been successfully applied both at the global and local level, both in academic institutions and in companies to solve problems requiring large computing power.

This research work is partially supported by the Network of Excellence CoreGRID (Contract IST-2002-004265) and CancerGrid (Contract LSHC-CT-2006-037559) funded by the European Commission under FP6 and by the Hungarian Jedlik Ányos HAGrid project (Grant No.: NKFP2-00007/2005).

1. Introduction

Desktop grids are an emerging trend in grid computing. Contrary to traditional grid [11] systems where the maintainers of the grid infrastructure provide resources as a service on which users can run their applications, maintainers of desktop grids provide the applications as a service and the participants of the desktop grid infrastructure provide the resources to run them. More specifically, in desktop grids we can distinguish participants who donate resources to the Grid (donors) and participants who utilise the resources to execute their computations (users).

Originally, the aim of Grid research was to realise the vision that anyone could donate resources for the Grid, and anyone could claim resources dynamically according to their needs, e.g. in order to solve computationally intensive tasks. This twofold aim has been, however, not fully achieved yet. Currently, we can observe two different trends in the development of grid systems, according to these aims.

Researchers and developers following the first trend are creating a service oriented Grid, which can be accessed by lots of users. To offer a resource for the Grid installing a predefined set of software (middleware) is required. This middleware is, however, so complex that it needs a lot of effort to install and maintain it. Therefore, individuals usually do not offer their resources this way but all resources are typically maintained by institutions, where professional

system administrators take care of the complex environment and ensure the high-availability of the Grid. Examples of grid infrastructures following this trend are the largest European Grid, the EGEE Grid [9] (comprising Virtual Organisations such as its Hungarian affiliate Virtual Organisation: HunGrid), or the NGS (National Grid Service [16]) in the UK. The original aim of enabling anyone to join the Grid with one's resources has not been fulfilled by this trend. Nevertheless, anyone who is holding a certificate from a trusted Certificate Authority of such a grid system can access the resources offered by the Grid.

A complementary trend can also be observed for realising the other part of the original aim. According to this direction anyone can bring resources into the Grid, offering them for the common goal of that Grid. The most well-know example of such systems, or better to say, the original Internet-based distributed computing facility example is SETI@home [1]. In Grids following the concepts of SETI@home, a large number of PCs (typically owned by individuals) are connected to one or more central servers to form a large computing infrastructure with the aim of solving problems with high computing needs. Such systems are commonly referred to by terms such as Internet-based Distributed Computing, Public-Resource Computing or Desktop Grids; we will use the term Desktop Grid (DG).

Offering resources to a DG is very easy: the owner just installs a simple client program package on her PC, registers herself on the web page of the DG and configures the client to offer resources to one or more projects supported by the Grid system. After this, the local software runs in the background or as a screen-saver to exploit otherwise idle cycles. After joining the Grid the owner does not need to take care of the Grid activity of her computer. Consequently, this makes DGs able to utilise a huge amount of resources that were not available for traditional grid computing previously. Applications can utilise resources in a DG by the well-known Master/Worker paradigm. I.e. the application is split up into many sub-tasks that can be processed independently (e.g. splitting input data into smaller, independent data units). Sub-tasks are then processed by the individual PCs, running the same worker executable but processing different input data. The central server of the Grid runs the master program, which creates the sub-tasks and processes the incoming sub-results.

The main advantage of DG systems is their simplicity for donors thus, realising the other part of the original Grid vision. However, as the complexity required to build a Grid must be handled somewhere, they also have disadvantages:

1. it is not so easy to setup and operate the servers required to run a DG system;

2. the application must be modified to run on the DG which might require fundamental modifications and redesign making the application specific to a particular DG infrastructure;

3. and the problems that can utilise a DG are also currently restricted to problems computable by the Master/Worker paradigm and have to be highly parallel since the clients computing the sub-tasks do not know about the other clients and cannot directly communicate with them.

Consequently, the enormous number of resources provided by donors is only accessible by a few users. This is shown by the limited number of projects utilising DGs currently on a world-wide scale to solve very large computational tasks such as search for extraterrestrial intelligence, high energy physics, cancer research, climate predictions, etc.

The next section discusses different DG-like systems that are in use today, the most widespread of them being BOINC. After that, SZTAKI Desktop Grid is introduced which aims to address the problems mentioned above and its most important features are described in detail, which help converge the two trends discussed earlier as an effort to bring the original aim and the Grid vision closer. The conclusion section presents some projects using SZTAKI Desktop Grid and summarises the paper.

2. Related work

2.1. BOINC

BOINC [2] (Berkeley Open Infrastructure for Network Computing), originated from the SETI@home project, is an effort to create an open infrastructure to serve as a base for all large-scale scientific projects that are attractive for public interest and having computational needs so that they can use millions of personal computers for processing their data. BOINC, in contrast to the original SETI@home distributed computing facility, can run several different distributed applications and yet, enables PC owners to join easily by installing a single software package (the BOINC Core Client) and then decide what projects they want to support with the empty cycles of their computers, without the need to delete, reinstall and maintain software packages to change between projects. BOINC is the most popular DG system today with the aggregated computational power of the more than 250.000 participants is about 475 TeraFLOPS thus, providing the most powerful "supercomputer" of the world.

2.2. Condor

While aiming for similar goals, Condor's approach is radically different from the DG concept. Condor [8] employs a push model by which jobs can be submitted into a

local resource pool or a global Grid (friendly Condor pools or a pool overlaid on Grids using different middleware). The DG concept on the other hand applies the pull model whereby free resources can call for task units. The scalability of Condor is limited by the centralised management implied by the push model (largest experiments are at the level of 10000 jobs in EGEE but it requires a very complicated Grid middleware infrastructure that is difficult to install and maintain at the desktop level). Condor provides a complex matchmaking feature to pair jobs and resources.

2.3. XtremWeb

XtremWeb [10] is a research project, which, similarly to BOINC, aims to serve as a substrate for Global Computing experiments. Basically, it supports the centralised set-up of servers and PCs as workers. In addition, it can also be used to build a peer-to-peer system with centralised control, where any worker node can become a client that submits jobs. It does not allow storing data, only job submission.

2.4. Commercial Desktop Grids

There are several companies providing a Desktop Grid solution for enterprises [12], [7], [5], [13]. The most well-known examples being Entropia Inc, and United Devices. Those systems support the desktops, clusters and database servers available at an enterprise. However, what they all have in common is that they run in isolation. There is no adherence to Grid standards and no interoperability amongst them or with other Grid middleware. They do not have the possibility to communicate with existing Grid installations other than on an ad-hoc basis. It is very likely they are also based on the push model.

3. SZTAKI Desktop Grid

Today, most of the DG projects (including SZTAKI Desktop Grid) utilise BOINC because it is a well-established free and open source platform that has already proven its feasibility and scalability and it provides a stable base for experiments and extensions. BOINC provides the basic facilities for a DG in which a central server provides the applications and their input data, where clients join voluntarily, offering to download and run an application with a set of input data. When the application has finished, the client uploads the results to the server. BOINC manages the application executables (doing the actual work) taking into account multiple platforms as well as, keeping a record of and scheduling the processing of workunits, optionally with redundancy (to detect erroneous computation either, due to software or hardware failures or clients being controlled by a malicious entity). Additionally, BOINC has support for user credits, teams and the web-based discussion forums, relevant in large scale public projects that are based on individuals donating their CPU time. These individuals must have a motivation for doing this. Apart from the project having a clearly stated, supportable and visionary goal, credits provide a kind of "reward" for the received CPU time, which leads to a competition between the users thus, generating more performance.

3.1. Local Desktop Grid

The advantages provided by the DG concept are not only useful on a world-wide scale but can also be used for smaller scale computations, combining the power of idle computers at an institutional level, or even at a departmental level. The basic building block of SZTAKI Desktop Grid is such a Local Desktop Grid (LDG) connecting PCs at the given organisational level. SZTAKI LDG is built on BOINC technology but is oriented for businesses and institutes. In this context it is often not acceptable to send out application code and data to untrusted third parties (sometimes this is even forbidden by law, as in the case of medical applications) thus, these DGs are normally not open for the public, mostly isolated from the outside by firewalls and managed centrally. SZTAKI LDG focuses on making the installation and central administration of the LDG infrastructure easier by providing tools to help the creation and administration of projects and the management of applications. LDG also aims to address the security concerns and special needs arising in a corporate environment by providing a default configuration that is tailored for corporate use and configuration options to allow faster turn around times for computations instead of the long term projects BOINC is intended for.

SZTAKI LDG is distributed prepackaged, so it can be easily installed using the `apt` tool on Debian GNU/Linux systems. After installation the `boinc_create_project` command can be used to create a new project. This creates everything needed for the project: a working directory, a database, an administrative user account and default configuration files for the web server and BOINC to make the project accessible. Administering is done using the administrative user of the project but for security reasons not by directly logging into it rather, acquiring the rights of this user when needed authenticating with their own password. The system administrator can grant or revoke project administrative rights to/from users via the `boinc_admin` tool. Project administrators are allowed to install application executables (master, client, validator), start/stop the project and access the database and administrative pages of the project. The `boinc_appmgr` tool can be used for automatic installation and configuration of packaged application binaries that come in an archive containing an XML description (provided by the application developer).

3.2. Hierarchical Desktop Grid

SZTAKI LDG can satisfy the needs of university departments and small businesses but what if there are several departments using their own resources independently but there is a project at a higher organisational level (e.g. at a campus or enterprise level). Ideally, this project would be able to use free resources from all departments. However, using BOINC this would require individuals providing resources to manually register to the higher level project which is a high administrative overhead and it is against the centrally managed nature of IT infrastructure within an enterprise.

A feature of SZTAKI Desktop Grid provides a solution to this: the possibility to build a hierarchy of LDGs. In a hierarchy, DGs on the lower level (child) can ask for work from higher level (parent). If a basic LDG is configured to participate in a hierarchy, the server can enter a hierarchical mode, when its clients require more work than it has for disposal. When the child node has less work than resources available, it will contact a parent node in the hierarchical tree and request work from it. The BOINC framework was not prepared for this functionality, so it had to be enhanced.

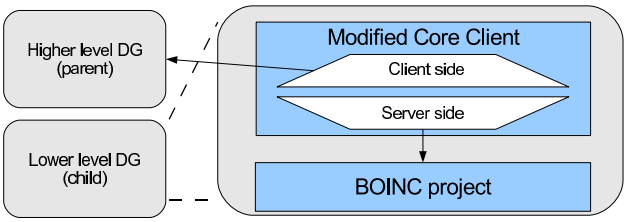

Figure 1. Hierarchy client

Hierarchical mode is implemented by a hierarchy client, which is run on the child LDG server. This way, the parent does not have to be aware of the hierarchy, it sees the child as one powerful client. The hierarchy client has two sides (see Figure 1): a master side which puts retrieved workunits in the database of the LDG and gets the computed results, and a client side which retrieves workunits from the parent and uploads results. The client side is a modified version of the standard BOINC Core Client. Modifications include:

- reporting a preconfigured number of processors to the parent independent of the actual number of processors on the local host, to allow it to request work for all its clients;
- reporting a preconfigured operating system and hardware architecture (Platform in BOINC terminology) allowing it to identify itself each time with a different Platform, so that Client Application executables for all clients could be obtained.

More detailed description of the hierarchical SZTAKI Desktop Grid can be found in [14].

3.3. DC-API

The SZTAKI Desktop Grid is based on BOINC thus, applications using the BOINC API can run on it. However, a simpler and easier-to-use API, the Distributed Computing Application Programming Interface (DC-API), is provided. The DC-API is the preferred way for creating applications for SZTAKI Desktop Grid. It aims to be simple and easy to use. Just a few functions are enough to implement a working application, but there are additional interfaces in case the application wants greater control or wants to use more sophisticated features of the grid infrastructure. Another purpose of the DC-API is to provide a uniform interface for different grid systems. It allows easy implementation and deployment of distributed applications on multiple grid environments. To move an application using the DC-API from one grid infrastructure to the other, it only needs to be recompiled with a different DC-API backend; the source code does not have to be modified. This enables scientists to concentrate on the application logic without having to know the details of the grid infrastructure or even know what grid infrastructure is serving their processing needs.

DC-API backends exist to use the Condor job manager and BOINC as well as a backend for the Grid Underground middleware used by the Hungarian ClusterGrid Initiative [6]. A simple fork-based implementation that runs all workunits on the local host is also available. The ability of running the workunits locally makes application debugging easier. Since switching the application from using such a local implementation to e.g. BOINC needs only a recompilation without any changes to the source code, the complete application can be tested on the developer's machine before deploying it to a complex grid infrastructure.

To accommodate the restrictions of different grid environments and to facilitate converting existing sequential code written by scientists not comfortable with parallel programming, the DC-API supports a limited parallel programming model only. This implies the following restrictions compared to general parallel programming:

- Master/Worker concept: there is a designated master process running somewhere on the grid infrastructure. The master process can submit worker processes called workunits.
- Every workunit is a sequential application.
- There is support for limited messaging between the master and the running workunits. However, this it is not suitable for parallel programming, it is meant to be used for sending status and control messages only.
- No direct communication between workunits.

Following the Master/Worker model, DC-API applications consist of two major components (see Figure 2): a

master and one or more client applications. The master is responsible for dividing the global input data into smaller chunks and distributing them in the form of workunits. Interpreting the output generated by the workunits and combining them to a global output is also the job of the master. The master usually runs as a daemon, but it is also possible to write it so it runs periodically (e.g. from *cron*), processes the outstanding events, and exits. Client applications are simple sequential programs that take their input, perform some computation on it and produce some output.

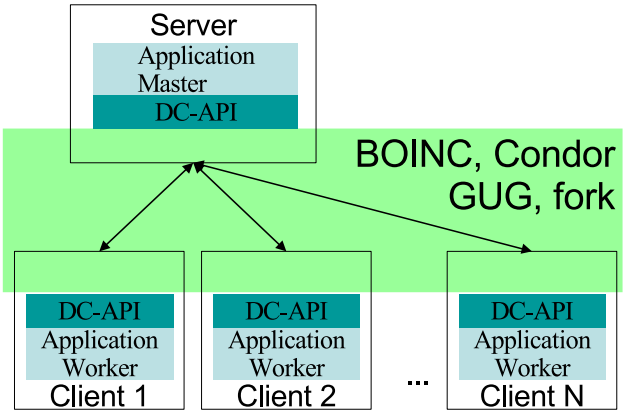

Figure 2. DC-API application components

A typical master application written using DC-API does the following steps:

1. Initialises the DC-API master library by calling the `DC_initMaster` function.
2. Calls the `DC_setResultCB` function and optionally some of the `DC_setSubresultCb`, `DC_setMessageCb`, `DC_setSuspendCb` and `DC_setValidateCb` functions, depending on the advanced features (messaging, subresults, etc.) it wants to use.
3. In its main loop, calls the `DC_createWU` function to create new workunits when needed and after specifying the necessary input and output files (`DC_addWUInput`, `DC_addWUOutput`) it can hand them over to the grid infrastructure for processing by calling the `DC_submitWU` function. If the total number of workunits is small (depending on the grid infrastructure), then the master may also create all the workunits in advance. If the number of workunits is large, the master may use the `DC_getWUNumber` function to determine the current number of workunits processed by the grid infrastructure, and create new workunits only if it falls below a certain threshold.
4. Also in its main loop the master calls the `DC_processMasterEvents` function that checks for outstanding events and invokes the appropriate callbacks. Alternatively, the master may use the `DC_waitMasterEvent` and `DC_waitWUEvent` functions instead of `DC_processMasterEvents` if it prefers to receive event structures instead of using callbacks.

A typical client application performs the following steps:

1. Initialises the DC-API client library by calling `DC_initClient` function.
2. Identifies the location of its input/output files by calling the `DC_resolveFileName` function. Note that the client application may not assume that it can read/create/write any files other than the names returned by `DC_resolveFileName`.
3. During the computation, the client should periodically call the `DC_checkClientEvent` function and process the received events.
4. If possible, the client should call the `DC_fractionDone` function with the fraction of the work completed. On some grid infrastructures (e.g. BOINC) this will allow the client's supervisor process to show the progress of the application to the user. Ideally the value passed to this function should be proportional to the time elapsed so far compared to the total time that will be needed to complete the computation.
5. The client should call the `DC_finishClient` function at the end of the computation. As a result, all output files will be sent to the master and the master will be notified about the completion of the work unit.

3.4. Clusters and Parallel Applications

As mentioned above, DGs are currently restricted to be used by applications following the Master/Worker paradigm. One of the applications of SZTAKI Desktop Grid however, required to overcome this limitation to provide a Grid execution environment for numerical weather prediction and climate models.

ALADIN is a modern finite domain numerical weather prediction model and complex meteorological model family. As such, the program code of ALADIN is quite large and complex. Moreover, as many scientific applications of this scale, it is using MPI for message-passing parallelism and consists of tens of thousands lines of FORTRAN. The first step in porting this application to desktop grids was the analysis of the communication patterns of the application. This has shown that ALADIN requires intense communication between its processes as their computation is tightly coupled and need each other's intermediate results to continue their own computation. Considering all of this, extending the DG infrastructure to allow running applications

without requiring them to directly employ DC-API and supporting cluster resources instead of single PCs seemed to be a much easier solution than re-engineering ALADIN itself.

For this, another modified BOINC client and a *wrapper* application was developed. The wrapper is wedged between the client and the real application, doing the required steps on behalf of the application and executing the parallel program on the client cluster (see Figure 3). This way, it became possible to run applications (such as ALADIN) on LDG without modifying their source code, where this is not feasible because of its complexity and size.

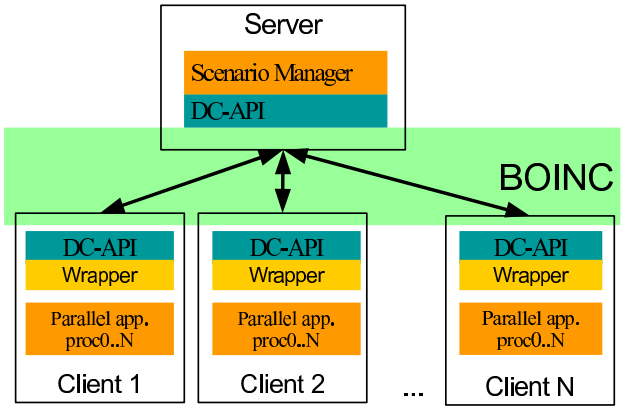

Figure 3. Components of Cluster and Parallel Application Support

For proper operation it is required that the parallel application (ALADIN) is registered in the database of the server, and that the runtime environment required by the application (such as MPI) is available on the clients. In a centrally managed LDG this can be assured. The input of the application must be divided into independent chunks which are to be processed by the parallel application running on the DG system. As this is dependent on the application this step should be done by scientists familiar with the application.

Another component which had to be implemented is the Scenario Manager. This component can be implemented using DC-API and has to perform the following tasks:

1. Having the independent chunks of input, it should create the appropriate workunits (`DC_createWU`) and submit them to the grid infrastructure for processing (`DC_submitWU`). In Figure 4 arrows 1 and 2 correspond to this task.

2. Handle the processed results (`DC_getResultWU`) and create the appropriate output files (`DC_getResultOutput`). In Figure 4 arrows 3 and 4 correspond to this task.

Creating the chunks of input and assembling the final output from the parts computed on the DG infrastructure can either be done by pre- and post-processing steps or be incorporated into the Scenario Manager.

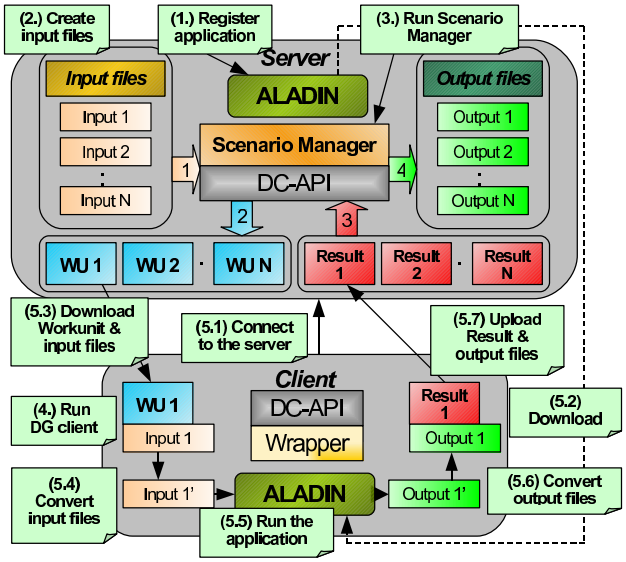

Figure 4. Flow of Wrapped Parallel Application Execution

During operation the following steps are performed:

(5.1) The client connects to the server (more precisely to the appropriate project on it).

(5.2) It downloads the application executable if it was not done yet. (The executable is only downloaded once by each client thus, network traffic is only dependent on the size of the input and output files.)

(5.3) It downloads a workunit (with all its input files) corresponding to the parameters of the cluster.

(5.4) After that, the client starts the wrapper which is part of the modified core client. The wrapper resolves all file names using the appropriate DC-API calls.

(5.5) The wrapper starts the real application executable (ALADIN) and waits for it to finish its execution.

(5.6) After the application has executed the wrapper restores the output files as it is required by the DG infrastructure and then finishes its execution.

(5.7) The Core client uploads the output files and reports the result to the server

After the above, the Scenario Manager can get the result from the infrastructure and extract the necessary output files using the appropriate DC-API functions, and can save them for later processing. When all submitted workunits have

been processed it can assemble the final output from the results if needed.

The wrapper works as a DC-API client application, i.e., at start up it calls the `DC_initClient` function, then reads the contents of the `native_wrapper.config` configuration file. This configuration file is created by the modified Core client from data found in the result descriptor of BOINC and describes the files needed by the real application executable. Then after resolving their real names, the wrapper copies the application binaries and the required input files into the working directory and starts the executable designated in the configuration file. After the application has finished its execution, the wrapper copies the output files declared in the configuration file to the paths required by BOINC and calls `DC_finishClient`.

Note that BOINC requires the application to create all its declared output files. Since the aim of this modified client and the wrapper is to allow any *non* gridified application to be run on LDG, this has to be solved. Because of this, the wrapper creates an empty file for each declared output file that cannot be found in the working directory after successful execution of the application (i.e. if some output files are optional for the application). However, as an exception this does not concern files with a name starting with `dc_`. These files are private to DC-API and should not be touched by the wrapper.

4. Conclusion

Several projects are actively using SZTAKI Desktop Grid and its features discussed in this paper. SZTAKI also runs a public BOINC project also known by the name SZTAKI Desktop Grid (SZDG) which is using DC-API and features of LDG at a global level.

The original intention of SZDG was to serve demonstrational purposes among scientists mainly in Hungary and also worldwide to prove that it's not necessary to have expensive hardware and truly monumental aims to catch the attention of the voluntary public. Soon after starting SZDG in early summer of 2005, the Computer Algebra Department of the Eötvös Loránd University applied for the project with their already developed and running single-threaded program: project BinSYS [4] which was modified to run on SZDG using DC-API. The goal of BinSYS was to determine all of the binary number systems up to the dimension of 11. The difficulty in this is that the number of possible number-system bases explodes with the rising of the dimension. The input of the program is a huge, but finite part of the number space and the output is a bunch of matrices, or more precisely their characteristic polynomials fulfilling certain criteria. Further narrowing the criteria on these selected matrices, the resulting ones make up the generalised binary number systems of the given dimension. Knowing the complete list of these number systems the next step is to further analyse from the view of information theory. Sketching the integer vector of the vector space in the usual way and in the generalised number system, their form can greatly vary in length. Also the binary form of vectors close to each other can vary on a broad scale. With this in mind the research will continue further with the use of number systems in data compression and cryptography. The assumption was that the program will be able to handle the dimensions up to 11 on a few computers and by the time the cooperation has been started it has already finished up to dimension 9. The prediction has assumed that the processing of dimension 10 will last for about a year, yet it has been successfully finished by the end of the year 2005 with the help of the few thousand computers of volunteers joining the project. After this success the application has been further developed making it able to handle dimensions higher than 11 and also to break the barriers of the binary world and process number systems with a base higher than 2.

One of the fundamental tasks in modern drug research is the exclusion of chemically unstable, biologically inactive or toxic compounds from the research process in the early stages, thereby reducing the cost and the time period of the drug development. The main purpose of the ADMETox-Grid and CancerGrid projects is to develop an enterprise Grid system that is suitable for predicting these chemical parameters of millions of compounds in a short time and in a secure manner, while also exploiting the free capacity of the office computers located at the different sites of the company. In this project SZTAKI Desktop Grid serves as the base of the Grid framework and is being extended with additional features such as cluster support, user interface via a web portal and advanced interface to databases.

In cooperation with University of Szeged, research is underway in the field of data mining and artificial intelligence to develop an algorithm utilising the capabilities of desktop grids. The software enables the user to select the algorithm and to make scheduling decisions while the algorithm is running, as well as the generation of higher quality data mining models by automating these permits. One of the most innovative parts of the research is to optimise the scheduling of the algorithms enabled by meta-level learning. A prototype running on the Local Desktop Gird set up at the University of Szeged supports the documentation and verification of data mining projects, while it remains expandable thanks to its architecture. Special attention is paid to data privacy issues. After the termination of the project, the prototype and its subsequent versions will be available for non-profit research purposes. Following up on the results of the project, the members will consider the commercial deployment of a data mining grid-based product.

One of the goals of the Hungarian Advanced Grid (HA-Grid) project is to elaborate a new generation of Desktop

Grids based on the achievements of SZTAKI Desktop Grid and to provide a Grid execution environment for numerical weather prediction and climate models in cooperation with the Hungarian Meteorological Service. Participants are about to create a so-called Global Desktop Grid (GDG) environment in Hungary, which is the first attempt to apply the DG technology not only for academic/research purposes, but on the intranet infrastructure of companies. In the project, a GDG system is being built involving large amount of computational resources from three sites: SZTAKI, Hungarian Meteorological Service and econet.hu. Later, this GDG will be the prototype for a national GDG service which aims to integrate home PC owners, whose interest will be challenged in financial manner. This way, a service provider based Hungarian Grid market will be born, which can lead to a new kind of Internet service in the long-term.

After SZTAKI Desktop Grid has received international recognition, the second international success was the West-Focus GridAlliance between Brunel University and the University of Westminster which is dedicated to raising the profile of Grid computing in the West London region and to facilitate real Grid-solutions in the industry. One of their application deals with designing periodic non-uniform sampling sequences for digital alias free signal processing. This is a computationally intensive problem, in which single computer based solutions could easily run for days or even weeks. To reduce computation time, the sequential algorithm had to be parallelised, making it possible to execute parts of the calculations on different nodes of computational Grids at the same time. This in turn reduces the overall runtime of the application. The SZTAKI Desktop Grid based version of the DSP application has been demonstrated in 2006/Q1 with 100 PCs located at the two Universities in London. The runtime of 1 month of the algorithm running on a single PC has been impressively reduced to two days.

As a summary we can state that SZTAKI Desktop Grid (SZDG) can be applied at several levels. At the global level it has been working for more than a year attracting more than 13,000 participants donating more than 23,000 desktop machines. The typical performance of SZDG varies between 600-800 GFLOPS but it already achieved 1.5 TeraFLOPs peak performance. At the local level SZDG is also successfully used at several universities and in companies. The main attraction of the local SZDG is its capability of creating hierarchical DG systems as well as including clusters as clients. These extensions of the local SZDG compared to BOINC makes SZDG an excellent choice to build enterprise and institution DG systems from smaller local SZDG systems built at the department level. Institutions and enterprises often possess clusters that can be also connected into the local SZDG significantly increasing its performance. The hierarchical DG concept requires much more applications to run on DG systems and hence we have put significant effort to facilitate the adaptation of existing applications to the DG concept. The DC-API developed for SZDG provides and easy-to-use function set by which master/worker applications can be easily created and run in SZDG. More than that, the DC-API is generic enough to run the same master/worker application on different Grid systems. Overall, SZDG and DC-API concepts together enable - the easy and non-expensive creation and maintenance of local enterprise/institution Grid systems - fast creation and execution of DG applications. More information on SZDG and DC-API can be found at the SZTAKI web site [15].

References

[1] D. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. Seti@home: An experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, November 2002.

[2] D. P. Anderson. Boinc: A system for public-resource computing and storage. In *Proc. of 5th IEEE/ACM International Workshop on Grid Computing*, Pittsburgh, USA, November 2004.

[3] F. Berman, A. Hey, and G. Fox, editors. *Grid Computing - Making the Global Infrastructure a Reality*. John-Wiley & Sons, Ltd., 2003.

[4] Project binsys. `http://compalg.inf.elte.hu/projects/binsys/`.

[5] A. A. Chien. Architecture of a commercial enterprise desktop grid: the entropia system. In Berman et al. [3], chapter 12.

[6] Hungarian ClusterGrid Infrastructure Project. `http://www.clustergrid.niif.hu/`.

[7] P. Computing. Platform LSF. `http://www.platform.com`.

[8] T. T. D. Thain and M. Livny. Condor and the grid. In Berman et al. [3], chapter 11.

[9] EGEE Enabling Grids for E-SciencE. `http://www.eu-egee.org`.

[10] G. Fedak, C. Germain, V. Néri, and F. Cappello. Xtremweb: A generic global computing system. In *Proc. of CCGRID2001 Workshop on Global Computing on Personal Devices*. IEEE Press, May 2001.

[11] I. Foster. *The Grid: Blueprint For a New Computing Infrastructure*. Morgan Kaufmann, Los Altos, CA, 1998.

[12] Grid MP, United Devices Inc. `http://www.ud.com`.

[13] I. D. Inc. DeskGrid. `http://www.deskgrid.com`.

[14] A. Cs. Marosi, G. Gombás, and Z. Balaton. Secure application deployment in the hierarchical local desktop grid. In *Proc. of DAPSYS 2006 6th Austrian-Hungarian Workshop on Distributed and Parallel Systems*, Innsbruck, Austria, September 2006.

[15] SZTAKI Desktop Grid. `http://www.desktopgrid.hu/`.

[16] The UK NGS. `http://www.grid-support.ac.uk`.