Accepted Manuscript

Efficient extension of gLite VOs with BOINC based desktop grids

Ádám Visegrádi, József Kovács, Peter Kacsuk

PII: S0167-739X(13)00229-X
DOI: http://dx.doi.org/10.1016/j.future.2013.10.012
Reference: FUTURE 2477

To appear in: *Future Generation Computer Systems*

Received date: 23 August 2012
Revised date: 28 September 2013
Accepted date: 4 October 2013

Please cite this article as: Á. Visegrádi, J. Kovács, P. Kacsuk, Efficient extension of gLite VOs with BOINC based desktop grids, *Future Generation Computer Systems* (2013), http://dx.doi.org/10.1016/j.future.2013.10.012

This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.

Highlights (for review)

ACCEPTED MANUSCRIPT

- The EDGI project aims to integrate Service Grid and Desktop Grid systems.

- MetaJob support enables the forwarding of multiple jobs to Desktop Grids through Service Grid systems

- The MetaJob provides seamless job forward since the service grid system does not recognise it as multiple job.

- Seamless job forward is the key to the performance issues caused by the submission of high-number of jobs.

- MetaJob can be used in any current service grids like gLite, ARC, UNICORE.

ACCEPTED MANUSCRIPT

*Manuscript
Click here to view linked References
ACCEPTED MANUSCRIPT

Efficient extension of gLite VOs with BOINC based desktop grids

Ádám Visegrádi, József Kovács*, Peter Kacsuk

MTA SZTAKI, Kende u. 13-17, 1111 Budapest, Hungary
{a.visegradi, smith, kacsuk}@sztaki.hu

Abstract

The paper describes the results of the EU FP7 EDGI project concerning how to extend gLite VOs with public and institutional BOINC desktop grids. Beyond simply showing the integration architecture components and services, the main emphasis is on how this integrated architecture can efficiently support parameter study applications, based on the so-called metajob concept created by the EDGI project. The paper explains in detail how to use the metajob concept by gLite users to exploit the BOINC desktop grids connected to the gLite VO, as well as how metajobs are managed internally by the 3G Bridge service. Performance measurements show that the Metajob concept indeed can significantly improve the performance of gLite VOs extended with desktop grids. Finally, the paper describes the practical ways of connecting BOINC desktop grids to gLite VOs and the accounting mechanism in these integrated grid systems.

Keywords: gLite, boinc, volunteer computing, desktop grid, job submission, batch

1 Introduction

Service grid (SG) systems based on gLite [1], UNICORE [2], Globus [3] and ARC [4] middleware are intensively used to solve parametric applications where the same code is executed with a large number of different parameter systems. The maintenance of such a grid system is quite expensive since they typically use managed clusters as computing element resources. A much less expensive alternative to solve such parametric applications would be the use of desktop grid systems, where—instead of managed clusters—the spare time of desktop machines is used as computational resource. Typical examples of such desktop grid (DG) systems are BOINC [5], XtremWeb [6] and Condor [7]. The advantage of DG systems is that they need much less initial investment than grid systems. For example, in case of BOINC, a small server is enough to manage a very large number of desktop resources, and these desktop resources could be existing desktop machines of an institute or volunteer home computers. None of them generates significant cost for the organization that maintains the BOINC server and runs the BOINC project. Therefore, at a fraction of the cost, much larger number of computing resources can be collected in a DG system than in a SG system.

Of course, it would make no sense to throw away the existing SG systems and rebuild the infrastructure on a DG basis. Rather, the EU FP7 EDGeS and EDGI projects [8][9] proposed to maintain the existing SG systems and extend them with the extremely cheap DG resources. The extension should be as easy and as transparent as possible both for the users and for the VO administrators. In the current paper we describe in detail how this objective is achieved for the gLite → BOINC integrated systems. EDGI provides solution for extending other SG systems (ARC and UNICORE) with another DG system (XtremWeb) but these are not explained in this paper. The interested reader can find details of those solutions at the EDGI project web page [8] or in related publications [10][11].

During the EDGeS project the extension of gLite VOs with DG system has been solved, but the performance was not as good as we (and the user communities) expected. We had to understand the limitations of the EDGeS implementation in order to significantly improve the technical solution of the

ACCEPTED MANUSCRIPT

integration, and to make it really usable for the gLite user communities. The main limitations were as follows:

1) Desktop grids are really helpful when the same application should be executed on many different data sets. The bigger the parameter study the more advantageous is using the supporting desktop grid system. As a consequence, we had to realize that for solving individual job based applications the usage of desktop grids is perfectly useless. On the other hand the use of desktop grids is extremely beneficial if large parameter sweep applications should be executed. In practice, the majority of the current gLite applications belong to this class, so we decided to concentrate on the support of parameter sweep applications in EDGI. At this point we quickly realized that for such applications the gLite Workload Management System (WMS) becomes a bottleneck. It is not by chance that the gLite system is extended with pilot solutions in order to overcome this bottleneck problem. So we had to come up with a solution that reduces the WMS work the same way as the pilot mechanism and, at the same time, matches the concept of BOINC desktop grids.
2) Another problem that gLite users did not like in the EDGeS solution was that they had to port their application from gLite to BOINC. Although we had created tools like DC API [12] and GenWrapper [13] that significantly reduced the required porting effort, still the users were reluctant to port their applications from gLite to BOINC. Therefore, we had to introduce a solution that completely eliminates the need of this porting activity. For this purpose we have invented the GBAC (Generic BOINC Application Client) that, with the help of virtualization, solves this problem as described in [14].
3) The third major hurdle for the gLite users was that sometimes they had to wait for unexpectedly long time to get all the results of the parameter sweep applications. This is due to the tail effect issue of volunteer desktop grids [15]. The tail effect problem comes from the unreliable nature of the volunteer client machines of the BOINC system. For example, if a client takes a data set but is turned off for a long time, then the final result of the parameter sweep application is also delayed. Due to the tail effect, typically the last 10% of the jobs are executed nearly as long as the first 90% of the jobs. In order to avoid the tail effect problem one option is to use dedicated cloud resources, as it will be described in a forthcoming paper. There are some other possibilities that can be used without applying cloud resources. These techniques will be described in this paper.
4) Finally, the data transfer between the gLite data resources and the BOINC clients was a major bottleneck in EDGeS infrastructure. In this paper we also show how this problem can be solved.

Since the solution of Problem 2 was already published in a previous paper [14], and Problem 3 will be discussed in a forthcoming paper, in the current paper we show the solutions for Problem 1 and 4. We particularly focus on the solution of Problem 1, and explain how the Metajob concept is able to provide a solution for Problem 3 and 4, too.

The next question is: how easy or difficult is it to physically extend the gLite VO infrastructures with BOINC desktop grid resources? We show two alternative solutions. One possibility is to connect the gLite VO to an existing volunteer BOINC desktop grid. The other option is to set up a new desktop grid and connect it to the gLite VO. The paper shows that both solutions are easy; so, even from this point of view, there is no obstacle to extend and use gLite VOs with BOINC systems.

The important message of this paper is that all the major obstacles of utilizing BOINC desktop grids in gLite VOs are eliminated from both the users' and the infrastructure's point of view. From now on, any gLite VO can easily be extended with BOINC desktop grids, and users can easily and efficiently exploit this integrated infrastructure. There are two typical scenarios in extending gLite VOs by desktop grids. In the first case, the EDGeS@home public volunteer project can be attached to any gLite VO through the CREAM computing element that is being operated by EDGI (and by IDGF-SP). This scenario requires no effort from the gLite VO admin. In the second scenario a new local campus-wide desktop grid site is deployed in order to collect the resources in a university. To help deploying a desktop grid server a documentation site is provided by IDGF-SP where installation manuals and images with pre-deployed components can be found and utilized.

ACCEPTED MANUSCRIPT

After these improvements of the integrated gLite-BOINC system, EGI [16] also considers this infrastructure as an officially supported infrastructure type. However, to make this happen, EGI required applying the same monitoring and accounting infrastructure for BOINC desktop grids that is used for the gLite infrastructure. Therefore, we have developed the required monitoring and accounting infrastructure that will be described in Section 5.

The paper is organized as follows. Section 2 introduces the main concept of the gLite BOINC integration, the related infrastructure components and the mechanism behind. Section 3 continues with the details on job submission, monitoring and result retrieval mechanism. In Section 4 we show the results of our performance measurements. Section 5 describes the monitoring and accounting concept and its implementation for the BOINC systems integrated with gLite VOs. Section 6 is about the related work. Finally, we conclude in Section 7 on the work described in this paper.

2 Extending gLite VOs with BOINC systems

The goal of extending gLite VOs with BOINC systems is to transfer parametric jobs from the gLite VO to one or more supporting BOINC systems, and to distribute the large number of job instances of the parametric job among the large number of BOINC client resources.

In order to extend gLite VOs with BOINC systems, three new concepts were introduced. First of all, the security system of gLite and BOINC are so different that somehow this had to be harmonized by developing a joint security concept that is acceptable for both systems. Second, the CREAM computing element of gLite VOs had to be modified in order to handle the new, unified security concept. Finally, the job submission mechanism is also different in the two systems, so we needed a bridge that transforms gLite jobs into BOINC workunits. The generic architecture of extending gLite VOs with a BOINC system is shown in Figure 1.

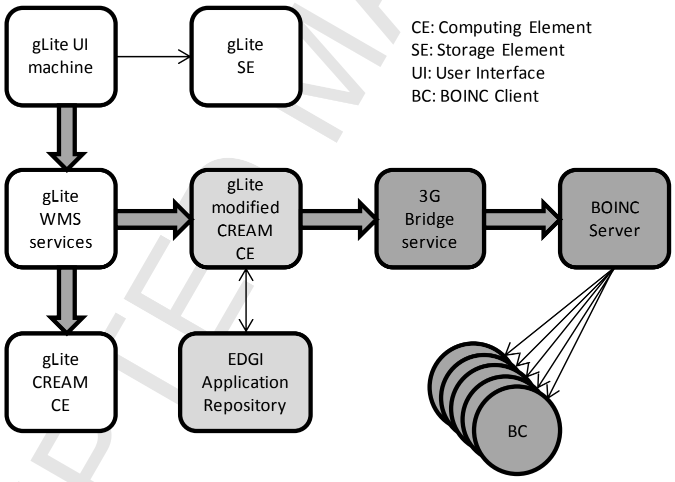

Figure 1 - EDGI Infrastructure to bridge gLite jobs to BOINC DGs

As Figure 1 shows, the parametric job submission goes through the following architecture services: gLite UI → gLite WMS → gLite mCE → 3G Bridge → BOINC Server → BOINC Clients (BCs).

Notice that the gLite VO system administrator does not need to modify anything in the original gLite VO. The only action that is needed in the gLite VO is to add to the existing VO a modified CE (mCE) that enables submitting jobs to the connected BOINC system.

These conceptual modifications and extensions of gLite VOs are explained in detail in the rest of the current chapter.

ACCEPTED MANUSCRIPT

2.1 Security system and EDGI Application Repository

Concerning the security mechanisms of SG systems, any user who has an accepted grid certificate can submit any kind of applications into the grid; i.e. the grid trusts the certified users. User certificates are not used in DG systems, DG clients trust the applications and not the users; and hence, only well tested and validated applications can be used in DG systems.

In order to combine the two concepts we have to restrict the applications that can be passed from the SG system to the supporting DG system. To achieve this, EDGI introduced the concept of application validation. Those parametric applications of the gLite VO that are intended to run in the supportive DG system should be validated before the infrastructure enables their transfer from the gLite VO to the DG system. By enabling the transfer, a mapping is realized between trusting the user and trusting the application. Transfer is only realized when a trusted user submits an application which is trusted by the target DG system.

To collect all the validated applications, a central Application Repository (AR) has been implemented and deployed. The EDGI AR [17] stores the validated/trusted applications with all of their executable binaries to be submitted.

2.2 Modified CREAM computing element

The CREAM computing element modified by the EDGI project is designed to forward jobs to desktop grid servers. The EDGI mCE has the following main tasks:

1) authorizing the application against the target DG system,
2) converting the application description to the format required by 3G Bridge,
3) keeping track of the status of the submitted job, and
4) retrieving the results.

In order to realize the situation, when a gLite job refers to a trusted application, we utilized the possibility of defining files by remote URLs. Therefore, instead of submitting (each of) the binaries to the mCE, users must only refer to the job binaries in the AR by gsiftp URLs. To help the reader understand how it is done, Figure 2 shows an example.

```
Executable = "dsp";
Arguments = "-f 22 -i 22 -p 723 -n pools.txt";
InputSandbox = {
      "gsiftp://edgi-repo.cpc.wmin.ac.uk:2811/srv/edgi/1001/1102/dsp",
      "pools.txt" };
OutputSandbox = {"result.txt", "stats.txt"};
ShallowRetryCount = 0;
RetryCount = 0;
SubmitTo = "cr1.edgi-grid.eu:8443/cream-pbs-edgidemo";
```

Figure 2 - Example JDL to submit an application to the DG through gLite

The example (to submit a single binary job named "dsp") shows that binaries must be referenced by URLs ("gsiftp://edgi-repo…") pointing to the EDGI AR inside the InputSandbox. The URLs for the binaries can be queried from the EDGI AR. These URLs are used by the EDGI mCE to detect whether the user refers to a trusted application stored in the AR. Input and output files can be defined in a normal gLite way, no restrictions are introduced. One additional and optional line ("SubmitTo") is added to make sure the gLite WMS will forward this job to the EDGI mCE in case the user wants to utilize desktop grid resources. However, the JDL is constructed in a way that it could be executed by a normal gLite CREAM CE.

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

Once the EDGI mCE has received the job and extracted all the necessary information (application name, arguments, inputs and outputs), submission to 3G Bridge [18] is performed through its WS interface. This interface is then used to keep track of the job status and to retrieve the result files.

The EDGI mCE is implemented as a new EDGI CREAM connector. The structure of the CREAM computing element allows attaching new connector components. A connector is for handling backend grids or clusters. The EDGI CREAM connector has the task to intercept the gLite jobs and to send them to a target 3G Bridge service. Thus, from higher level CREAM component's point of view, the new connector behaves like a batch system implementation. The difference is that the job is not run on a worker node belonging to the CREAM CE, but is sent to a 3G Bridge service for execution by a desktop grid system behind. A more detailed description of the EDGI CREAM computing element can be found in [19].

2.3 3G Bridge

The 3G Bridge (Generic Grid Grid Bridge) [18] is designed to be used as a mediator between different types of grid middleware. Its main goal is to realize a standard gateway between the various grid systems. It has three main parts:

5) web-service interface to receive the incoming jobs
6) database and queue manager to store and schedule the jobs
7) grid-handler interface and plugins realizing the interfaces to perform grid specific job handling

Regarding point 1), the web service interface ("WSSubmitter" in Figure 3) offers the most important job manipulation functionalities like submission, state query, result query, cancel, etc. The interface is used by the EDGI CREAM mCE. For 2), the bridge stores the job description in a relational database ("3G Bridge Job DB" in Figure 3) and the "Queue Manager" is responsible to invoke the different grid handlers to perform activities on a particular job. For 3), a "Grid plugin" implementing a "Grid Handler Interface" is responsible to communicate with the backend grid system like BOINC, XtremWeb, etc. The incoming jobs are organized into queues, and each queue must have a grid plugin handling the jobs of that particular queue. For example, a BOINC plugin takes jobs from a certain job queue and inserts them as workunits into the BOINC database. The EDGI mCE should submit its job to this particular queue in order to make sure the job will be transferred to BOINC. A more detailed description of the 3G Bridge service can be found in [18].

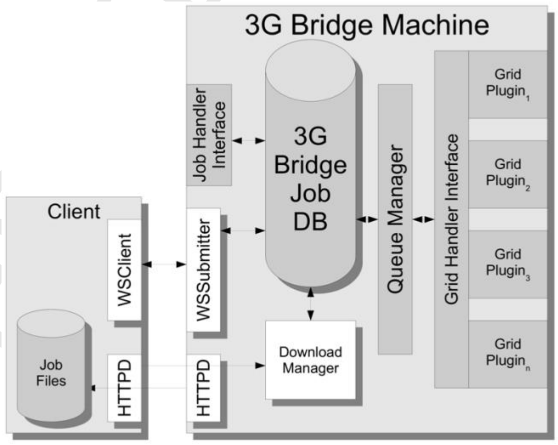

Figure 3 - Internal structure of the 3G Bridge service

2.4 Running parametric applications in the gLite-BOINC system

Although our main goal was to support parameter study applications, we have found inefficiencies of the system in this case.

ACCEPTED MANUSCRIPT

To submit parameter study applications, users may submit a gLite parameter study or job collection, which will be processed by the gLite WMS. The jobs in the collection will be handled—and possibly forwarded to the DG—individually. As the WMS has the liberty to send only a subset of the parameter study to the modified computing element, and because it does not know about the capabilities of the infrastructure behind the mCE, this approach cannot fully utilize the resources provided by the DG. Unfortunately, the situation is not much better even if the user explicitly specifies the mCE in the JDL as the destination CE. In this case, it is guaranteed that all the jobs of the parameter study application will be transferred to the connected DG system but jobs will be forwarded individually, and such an individual forwarding of the jobs in the PS will still have considerable overhead—as measured and described in Section 4.

3 Efficient submission using the Metajob feature

We have designed and implemented the *Metajob* feature, which enables gLite users to efficiently utilize the desktop grid for parameter study applications. Our design goals were the following:

1) As we are extending a working infrastructure, the new feature must affect the least components possible.
2) It must be easy to use.
3) The resulting solution must impose considerably less overhead on the infrastructure than existing solutions (described in Section 2.4).

To achieve our main design goal (3), our basic concept was that the user should submit a single package—a *metajob*—describing the actual jobs, which will be "unfolded" only when necessary. There are two components in the infrastructure that are completely under our control: the mCE, and the 3G Bridge. Being downstream in the flow of submission, we chose the 3G Bridge to be modified, because this way, we can shave off even the mCE–3G Bridge communication overhead.

We chose a solution that did not need the modification of any of the 3G Bridge interfaces; therefore, we could leave all other components unmodified—only the user needs to be conscious about using the feature.

The following subsections describe in detail how 1) and 2) have been achieved; while 3) is corroborated by the performance measurements in Section 4.

3.1 Submitting a parameter study

To submit a parameter study, the user has to submit a single, special job, a metajob, through the EDGI infrastructure. A metajob differs from an ordinary job in a single, specially named input file, which contains the description of the jobs in the parameter study. This file is called the *metajob file*. This special file will be noticed and interpreted only by the Bridge, which will unfold the metajob and execute the resulting *subjobs* in the desktop grid. The gLite infrastructure and the modified CE will see and handle the whole parameter study as a single job, which alleviates stress on the gLite infrastructure. Even more, the Bridge will not only create, but also *manage* all subjobs internally. It will aggregate their status information and their output files to fully hide the nature of the metajob from the gLite infrastructure. By doing so, the Bridge shifts further load from the gLite infrastructure to itself.

The JDL in Figure 4 can be used to submit a parameter study through gLite, to the desktop grid. Notice that the only difference between this JDL and a regular one (shown in Figure 2) is an extra input file called "*_3gb-metajob-test*". Although submitting this JDL will result in many jobs executed in the desktop grid, the gLite part of the infrastructure will handle it as a single one.

```
Executable = "dsp";
Arguments = "-f 22 -i 22 -p 723 -n pools.txt";
InputSandbox = {
      "gsiftp://dev17-portal.cpc.wmin.ac.uk:2811/srv/edgi/1001/1102/dsp",
      "pools.txt",
      "_3gb-metajob-test"};
```

ACCEPTED MANUSCRIPT

```
OutputSandbox = {"result.txt", "stats.txt"};
ShallowRetryCount = 0;
RetryCount = 0;
SubmitTo = "cr1.edgi-grid.eu:8443/cream-pbs-edgidemo";
```

Figure 4 - Example JDL to submit a parameter study to the DG through gLite

But how does a—technically—ordinary job become a batch of many jobs? The web-service interface of 3G Bridge would be able to recognize and unfold an incoming metajob upon submission, but this would introduce high delays when submitting: the remote procedure call would terminate only after the metajob has been unfolded, which time is proportional to the metajob's size. Also, the status and the output of the subjobs should be aggregated on demand, whenever a request arrives, which would make this solution non-scalable.

We have chosen to implement the Metajob feature as a backend plugin of the Bridge (Figure 5). The mCE submits the metajob to the Bridge queue assigned to the desktop grid; as it would do in case of an ordinary job. The web-service interface notices the extra input file—whose name begins with "_3gb-metajob"—, and redirects the job to another queue, so the Metajob plugin will handle it instead of the originally specified backend queue. The Metajob plugin will *unfold* the metajob first: it will interpret the definition file, and insert the subjobs in the 3G Bridge database. Then, the plugin will keep track of the subjobs, gather their output, and recursively cancel them when necessary.

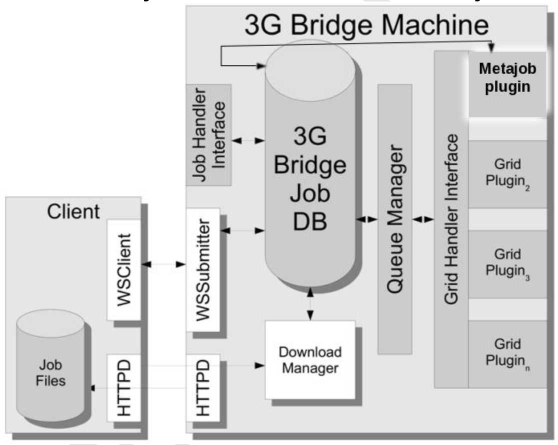

Figure 5 - 3G Bridge architecture with Metajob plugin

The extra metajob description file must contain the specification of the parameter study application. The parameter study can be specified using a simple imperative language (example shown in Figure 6). The language is based on the Condor job description language [7]. We have chosen this syntax as it is simple to write and to generate with scripts; and it can be parsed quickly with constant memory overhead (as opposed to an XML-based language).

The executable name, the set of input files, and the set of output files—that is, their logical names—are derived from the JDL, and are fixed. The user can set the arguments of a specific subjob (line 05 in the example of Figure 6), and the source URLs of each input file (lines 06, 12, 15); then, they can instantiate a subjob using the `Queue [N]` command (lines 07, 13, 16). The interpreter maintains a single job template, which is initialized based on the JDL, and can be manipulated with the assignments in the metajob description. Each `Queue [N]` command will instantiate subjobs based on the current state of the template. This means that unchanged values will be inherited by following subjobs.

Notice that input files are specified with URLs in the example. This is because the Metajob feature only allows http location specifications. The gsiftp protocol is not supported, since BOINC does not support it. Furthermore, since the infrastructure does not interpret metajobs, local (sandbox) files cannot be specified, because they would not be transferred with the metajob. Although this is a restriction, it

ACCEPTED MANUSCRIPT

encourages the use of the scalability improvement of the EDGI infrastructure and BOINC suggested by Problem 4 in Section 1.

Files specified with http URLs will only be downloaded as late as possible. If the URL is specified, only the 3G Bridge will download these files, so the gLite infrastructure will not have to transfer these files. Moreover, if the MD5 sum and the size of the input file is also specified (line 12), only the BOINC client will download the input file, completely removing transfer overhead from both the gLite infrastructure and the BOINC server.

The '%Comment' directive (lines 04 and 11) enables the user to identify which parameter set produced a particular result. A Comment may be specified for each set of parameters. All subjobs created by a single `Queue [N]` command will share the same comment, but—unlike other properties—it will not be inherited by following subjobs. The comment can be an arbitrary string, which will be included in the aggregated output to help find specific results. Using this feature is detailed in Section 3.4.

3.2 Controlling batch execution

As a metajob will be submitted as a single job, the infrastructure will only enable the user to control (monitor, cancel) the metajob, but not its individual subjobs. The subjobs may not execute uniformly: their execution time may have high variance, or some of them may fail. As the user has no control over individual jobs, this must be handled automatically by the Bridge, based on parameters specified by the user upon submission.

The user can specify two directives for the Bridge to handle the parameter study. The '%Minimum' and '%Maximum' directives (lines 01 and 02 in Figure 6) are properties of the parameter study, and control its execution and evaluation. The '%Minimum' directive specifies the *necessary* number of subjobs required for the parameter study to be successful. If less than this number of subjobs finish successfully, the whole parameter study is considered failed, and no output is produced. The '%Maximum' directive specifies the *sufficient* number of subjobs needed for success. If at least this number of subjobs has finished successfully, remaining running and pending subjobs are cancelled, and the execution of the parameter study is considered successful. Both values can be specified as absolute values or values relative to the total number of jobs—as percentage, or using the keyword 'All' standing for 100%.

This simple control feature supports important scenarios of parameter studies. For example, %Minimum=1 means that any result would be acceptable, while %Maximum<100% would introduce redundancy in the application.

The latter case may be used to reduce the tail effect in volunteer desktop grid systems (see Problem 3 in Section 1). Tail effect happens in case of job batches, when the variance in execution time is high. Some of the jobs may take exceptionally long time to execute, which delays the completion of the whole batch. In some cases—Monte Carlo simulations for example—discarding several results is acceptable; thus, it is possible to introduce redundancy. Introducing redundancy may reduce the execution time of the batch, as it would enable the Bridge to discard some or all offending workunits, so the batch can finish in less time.

```
File _3gb-metajob-test:
01    %Minimum 1
02    %Maximum 75%
03
04    %Comment With input file: par1.txt
05    Arguments = -f 22 -i 22 -p 723 -n pools.txt
06    Input=pools.txt = http://my.server.com/ps/par1.txt
07    Queue

08    # `Queue N` creates N identical subjobs. All of them will
```

ACCEPTED MANUSCRIPT

```
#  have the same %Comment.
# The Arguments property is inherited.
%Comment With input file: par2.txt
Input=pools.txt =
    http://my.server.com/ps/par2.txt=d8e8fca2dc0f896fd7cb4cb0031ba249=320
Queue 2

# %Comment is not inherited
Input=pools.txt = http://my.server.com/ps/par3.txt
Queue
```

Figure 6 - Example metajob description file

3.3 Monitoring the execution of the parameter study

As stated before, the Bridge fully hides the nature of a metajob from the infrastructure. To maintain this, the detailed status information of a parametric study must not be published through the infrastructure. Instead, the status of subjobs is aggregated by the Bridge, and only this aggregated status information will be available through the usual channels. The detailed information is made available to the user through an external channel.

Periodically, the statuses of the subjobs are gathered in a histogram. This histogram is used to generate a status file (see an example in Figure 7), which is published in a readable textual format, through http. This method requires no modification in the infrastructure and needs no extra maintenance since: (1) An http server is always present, because it is mandatory for BOINC. (2) The infrastructure already provides the internal DG identifier—gridid—of the job to the user. As a metajob has no internal DG identifier, this field can be recycled to hold the URL of the status information file.

The user can query the gLite logging information about the job from the WMS, which will contain an entry named ‘3GBridge_DG_ID’. In case of a metajob, this entry will contain the URL of the detailed metajob status information. An example status file is shown in Figure 7.

```
# Stat generated at Wed Apr 9 17:12:40 2013

Meta-job ID: 328808cc-7a66-4632-b3d8-0d6ef4c85c62
Meta-job STATUS: RUNNING

# Generation report
Total generated:    10500 (100.0%)
Required:           10000 ( 95.2%)
Success at:         10000 ( 95.2%)

# Status report
Not started:            0 (  0.0%)
Running:             1049 ( 10.0%)
Error:                  0 (  0.0%)
Finished:            9451 ( 90.0%)
Still need:           549 (  5.2%)
```

Figure 7 - Example of a detailed metajob status

The histogram of the subjobs’ statuses is then mapped to a possible job status that will become the overall status of the metajob; this mapped status will be reported to the modified CE. The mapping considers the Minimum and Maximum values specified in the metajob definition file. If the number of successfully finished subjobs exceeds Maximum, all remaining subjobs are cancelled, output is produced, and the metajob is reported to be successfully finished (”Finished”). If all subjobs have finished—either

ACCEPTED MANUSCRIPT

successfully or with error—the metajob is considered to be successful if and only if the Minimum number of successful subjobs has been reached. As an optimization, if the Bridge finds that so many subjobs have failed, that the Minimum cannot possibly be reached, remaining subjobs are cancelled, and the metajob fails immediately sending back the status “Error”.

In the example shown in Figure 7, if 501 subjobs would fail, then the whole metajob would be considered failed. On the other hand, after 549 subjobs have finished successfully, all reamining subjobs are cancelled immediately, and the metajob is considered successful.

3.4 Obtaining the output of the parameter study

As with status information, the infrastructure is only prepared to handle the results of the single submitted metajob. To make the infrastructure transfer all subjobs’ results back to the user, the Bridge must aggregate them, and pretend that the metajob has produced it.

To achieve this, for all files specified in the output sandbox, the Bridge will create a tar.gz archive, which will contain all corresponding output files of the subjobs. The name of the archive must match that specified in the JDL. For example, if the sandbox specifies a single output.txt, the metajob will produce a single result.txt. This file will actually be an archive containing all result.txt-s produced by successfully finished subjobs (Figure 8). If another file, for example stats.txt, is specified, the metajob will “produce” another archive named stats.txt containing all matching result of its subjobs, etc. To separate subjobs in an archive, they are put into separate directories; each directory name being the unique identifier of the corresponding subjob given by the Bridge. This way, to merge multiple output archives, one must simply decompress them in the same directory (Figure 9).

```
result.txt  # actually, a tar.gz with the following content
├── 0b/
│   └── 0b3e6bd3-f8b4-4d60-84a9-8c9c5855ffce/
│       └── result.txt
├── 26/
│   └── 263a6140-cf01-4889-ad21-210ecd3d41c4/
│       └── result.txt
└── 47/
    ├── 475ea5d0-982d-45d2-8260-7eb8493851e5/
    │   └── result.txt
    └── 4787ab3d-594f-4691-96db-b826c3d63b61/
        └── result.txt
```

Figure 8 - Example output of a metajob (result.txt)

```
# The result of uncompressing both result.txt
# and stats.txt in the same directory.
.
├── 0b/
│   └── 0b3e6bd3-f8b4-4d60-84a9-8c9c5855ffce/
│       ├── result.txt
│       └── stats.txt
├── 26/
│   └── 263a6140-cf01-4889-ad21-210ecd3d41c4/
│       ├── result.txt
│       └── stats.txt
└── 47/
    ├── 475ea5d0-982d-45d2-8260-7eb8493851e5/
    │   ├── result.txt
    │   └── stats.txt
    └── 4787ab3d-594f-4691-96db-b826c3d63b61/
        ├── result.txt
        └── stats.txt
```

Figure 9 - Merged output of a metajob

ACCEPTED MANUSCRIPT

The user must be able to tell which parameter set produced a specific result, but UUIDs are not useful for that. The Comment directive used in the metajob definition language was implemented to achieve this. In each output archive, the Bridge will also include a so called *mapping* file (Figure 10), which can be used to identify results. (The same mapping file is included in all output archives.) The mapping file is actually a metajob definition file, but it is “normalized”:

1. Relative specifications of Minimum and Maximum are changed to absolute values.
2. Inheritance is not used, all subjobs are fully defined.
3. Queue N commands are not used; they are substituted with N full subjob definitions.
4. Before each %Comment, a %Id directive specifies the UUID associated with that parameter set.

The mapping file is semantically equivalent with the original metajob definition. If submitted, the %Id directives will simply be omitted.

```
File _3gb-metajob-test-mapping:
%Id 0b3e6bd3-f8b4-4d60-84a9-8c9c5855ffce
%Comment With input file: par1.txt
Arguments = -f 22 -i 22 -p 723 -n pools.txt
Input=pools.txt = http://my.server.com/ps/par1.txt
Queue

%Id 263a6140-cf01-4889-ad21-210ecd3d41c4
%Comment With input file: par2.txt
Arguments = -f 22 -i 22 -p 723 -n pools.txt
Input=pools.txt = http://my.server.com/ps/par2.txt
Queue

%Id 475ea5d0-982d-45d2-8260-7eb8493851e5
%Comment With input file: par2.txt
Arguments = -f 22 -i 22 -p 723 -n pools.txt
Input=pools.txt = http://my.server.com/ps/par2.txt
Queue

%Id 4787ab3d-594f-4691-96db-b826c3d63b61
%Comment
Arguments = -f 22 -i 22 -p 723 -n pools.txt
Input=pools.txt = http://my.server.com/ps/par3.txt
Queue

%Minimum 1
%Maximum 3
# Total generated: 4
```

Figure 10 - A possible mapping file generated by submitting the metajob description file shown in Figure 6

The infrastructure will transfer the created archive output files back to the user like usual output files. The user only has to extract them, and, if looking for a particular result, check the mapping file for information. For example, if the user wants to find results belonging to par2.txt, they can look for “With input file: par2.txt” in the mapping file and find the UUIDs associated with these parameter sets. The UUIDs found will be the directory names containing the results sought.

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

We have developed the metajob feature in hope that it will reduce administrative and communication overhead considerably in the gLite part of the infrastructure. As the Bridge undertakes the responsibility of managing such parameter studies,

- the gLite infrastructure only has to forward a single job upon submission, reducing administration and transfer overhead;
- the gLite infrastructure only has to monitor a single job, reducing polling overhead;
- the user only has to manage a single job.

In case of submission, the metajob feature uses local database insertions instead of individual forwarding subjobs through the infrastructure. In case of monitoring and management, the most overhead comes from the polling nature of the infrastructure. If Metajob is used, only a single job has to be polled; while the Bridge can assemble the status histogram, required for subjob management, with a single database query. It is clear, that the strength of the metajob feature is that it shifts decision making to where the information exists.

Performance measurements have been executed to verify that the metajob feature achieves these goals. Our results will be described in Section 4.

4 Performance measurements

The key feature of the metajob concept is that it shifts administrative load from the WMS and the modified CE to the 3G Bridge. It also eliminates communication overhead between these elements. On the other hand, the architecture of the 3G Bridge, and the fact that it only has to process locally available information, makes it very scalable.

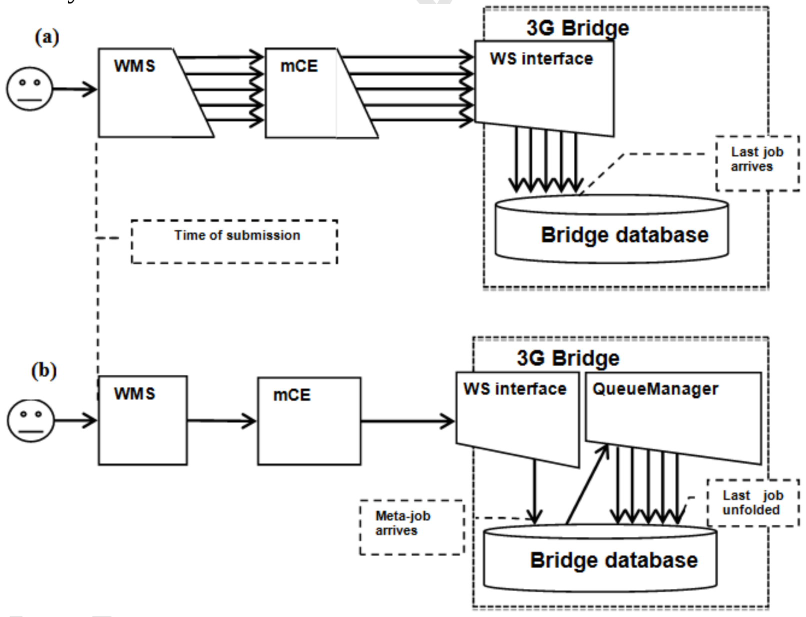

Figure 11 - The metric used in our measurements is the arrival time of all jobs in the batch (a) gLite collection, (b) metajob

We have designed and implemented a test to verify the efficiency of the metajob feature. The focus of the test was to measure the submission time of batches of jobs. That is, the time elapsed between submitting the batch to gLite, and all subjobs arriving in the Bridge, becoming ready to be forwarded to the desktop grid.

The alternative, to which we have compared the metajob feature, is gLite's collection submission. In both cases, a batch of jobs can be submitted to the EDGI infrastructure, and in both cases, the

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

infrastructure will manage the batch. The difference is that a gLite collection will be unfolded in the WMS, whereas a metajob will only be unfolded in the 3G Bridge.

The elapsed time, in both cases, is measured from the time of submission. In case of gLite collections, the time of arrival is considered to be the maximal time of arrival among subjobs; that is, when the WMS has successfully forwarded all subjobs to the Bridge. The equivalent metric in case of metajobs is to measure the time until the Bridge has finished unfolding the metajob (rather than until the metajob itself has arrived).

The measurements were executed on the EDGI infrastructure, backed up by the EDGIDemo desktop grid. In each test, a batch of 200*k ($k \in [1..5]$) identical jobs—based on the dsp executable (Digital Signal Processing application) — has been submitted to the desktop grid, through gLite, either as a gLite collection or a metajob. In case of metajobs, the metajob definition format would have allowed us to optimize the submission of identical jobs by using the Queue N command. As the gLite collection feature does not offer such optimization, we disregarded this possibility. On the other hand, as in the metajob case the proxy certificate has to be handled only once for each submission, we used the proxy delegation feature of gLite to simulate similar conditions for the collection case.

The time of arrival—in both cases—has been parsed from the Bridge logs; therefore, polling was not required. Log records are generated immediately when an event occurs, the timestamps are accurate to the second (no rounding), and no operations are performed (no cumulative error). Thus, the error is between (-1, 0] seconds. The median results—of five samples each case—are shown in Figure 12.

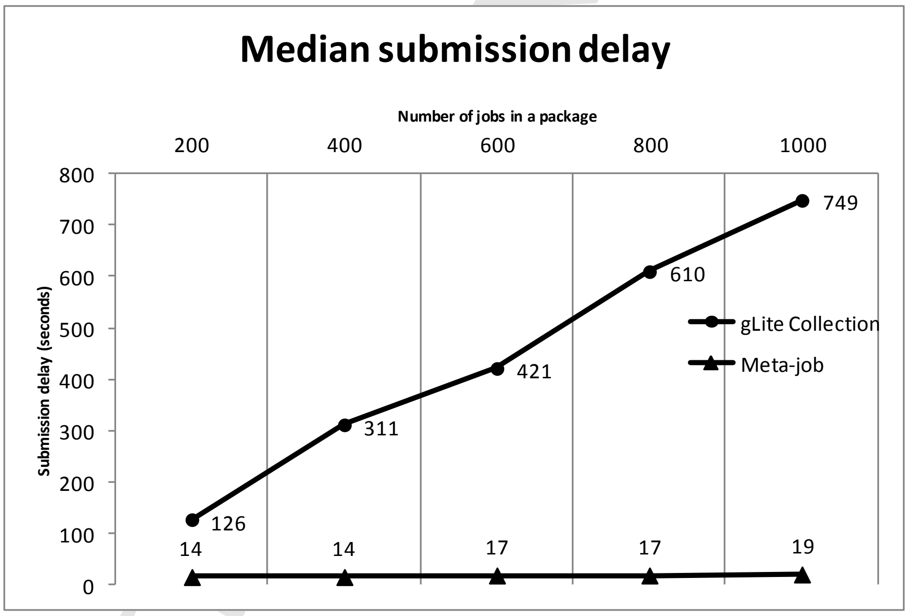

Figure 12 - Performance measurement results

The results match the expected behavior of the infrastructure. Suppose the time needed to forward a single job from the WMS to the Bridge takes t_F time. When a batch of n jobs is submitted as a gLite collection, it is unfolded immediately by the WMS, and then on, all subjobs are forwarded through the infrastructure, to the Bridge, individually. That is, forwarding the batch will take $\Theta(nt_F)$ time. In the metajob case, assuming that parsing a subjob and inserting it into the database takes t_p time, submitting a metajob with n subjobs will take $\Theta(t_F + nt_p)$ time. Considering that t_F consists of the administrative overhead at the infrastructure components *and* then transmission delay between them, while t_p depends only on parsing time and DBMS insertion time, t_F is expected to be several orders of magnitude higher than t_p. This essentially means that submitting a gLite collection should take linear time, while submitting

ACCEPTED MANUSCRIPT

a metajob should take quasi-constant time. The results of our experiment match this expectation perfectly: the submission time of a metajob grows at a negligible pace as more jobs are submitted in a batch.

5 Providing accounting information for the EGI federation

In order to fully integrate service grid infrastructure, three EU FP7 projects are collaborating: EDGI [8], EGI-Inspire [16], and EMI [20]. EDGI is working on integrating the SG and DG infrastructure together with EMI and EGI.eu. The SG–DG integration has three main logical parts:

1. Seamless transfer of jobs from gLite, ARC or UNICORE to BOINC or XtremWeb-based desktop grid sites. This task belongs to EDGI and this integration regarding job transfer has been successfully done by EDGI. As a result, modified computing elements have been developed and became part of the EMI software distribution. SLA and OLA has been signed between EDGI and EMI for further software support.
2. Monitoring desktop grid sites within the EGI monitoring infrastructure. This work has been done in the first half of 2012 within the framework of the EDGI–EGI MoU. EDGI has developed and later its follow-up FP7 project, called IDGF-SP maintains probes for monitoring the desktop grid. Probes are provided as RPM packages which follow EGI probe development guidelines. Currently, the Hungarian NGI is operating the Nagios probes for the DG sites.
3. Accounting for desktop grid sites for the EGI infrastructure. EGI maintains an accounting infrastructure based on APEL [21]. All EGI CEs must gather accounting information on executed jobs, and synchronize this information with the site-level APEL node. For full integration, the EDGI modified CE must also supply accounting information to the EGI system.

This section describes how the EGI accounting integration has been achieved in EDGI.

5.1 Required information

Most of the information required by the APEL infrastructure about jobs is available at the modified computing element. The actual accounting information about them, however, can only be provided by the desktop grid. The accounting metrics the desktop grid has to provide to the modified computing element are summarized in Table 1.

Start/stop times	Time when the job has been started, and when it has finished.
Wall clock time	Total time the job has spent running: = stop_time – start_time
CPU time	Consumed CPU time measured by the kernel on the executing host.
Memory	Real and virtual memory consumed by the job.
Benchmark values	Constant factors describing the performance of the executing host.
Number of CPUs	Number of CPUs in the executing host.

Table 1 – Accounting metrics provided by the desktop grid

Some of these metrics are readily available in the desktop grid database; however, there are three exceptions.

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

Memory

The memory consumed by the job is not recorded by the desktop grid. On the other hand, for each application, an upper limit for memory consumption must be estimated. This information is stored as an attribute for each workunit, and can be used by clients to filter out workunits that would require too much memory. If a running workunits exceeds its limit, it will be immediately terminated.

Because of the termination policy, it is guaranteed, that the memory consumed by the workunit will be at most that estimated. Because of the filtering policy, application developers are forced to provide tight estimates. Therefore, the limit associated with the workunits is a suitable estimate for the memory consumption of the job.

Benchmark metrics

Only the number of floating point operations per second (FLOPS) is provided as information about hosts' performance. The current APEL records store SpecInt2K and SpecFloat2K factors (although this seems to be changing [22]). To integrate the desktop grid with the APEL accounting system, we have to find a reliable mapping between these two values.

As the specification of APEL seems to be changing, the simplest and best solution is that the desktop grid will only provide the raw FLOPS value to the computing element. Then, the computing element will be able to map the raw value to a suitable number as necessary.

Benchmark metrics in case of metajobs

In case of metajobs, most of these metrics are trivial aggregations of the metrics of their subjobs. Start time is the minimal start time, stop time is the maximum stop time, while consumed memory, number of CPUs and CPU time can be summed.

However, determining benchmarking metrics is not trivial. The benchmark being the number of floating point (or integer) operations a specific host can perform in a second, simply summing the individual metrics of all the hosts would result in a skewed metric that does not take into account the effort (time) each host has made. The solution is the following simple metric.

For each host, the number of floating point operations it has actually performed can be estimated as follows: $FP_h = FLOPS_h * CPUtime_h$. This is a statistic that can be summed. The total CPU time donated by all the hosts can also be summed. Thus, the following statistic makes sense:

$$\frac{\sum_h FP_h}{\sum_h CPUtime_h}, \text{where } FP_h = FLOPS_h * CPUtime_h; \;\; h \in \text{Hosts}$$

This expression captures the overall, weighted performance of the set of nodes that has executed the metajob.

5.2 Propagating information

The modified computing element communicates with the desktop grid via the web-service interface of the 3G Bridge. The current interface did not define a way to access detailed information about a job; therefore, a new method had to be made available.

We have implemented a RESTful interface that is much simpler, can be accessed with standard http clients (browsers, curl, wget, etc.), supports authentication and authorization, and is very easy to extend with new functionality. The required function has been implemented and made available through this new REST interface.

The 3G Bridge itself cannot provide the accounting information needed, only its back-end desktop grid can. However, the desktop grid database and the 3G Bridge database are physically the same; therefore,

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

the 3G Bridge interface can access it and extract the required information from it directly. This means, that the interface between the 3G Bridge and the back-end desktop grid can be left intact.

The modified computing element must call this new function with a job identifier as an argument. As a result, the information specified in Table 1 is returned as a table of key–value pairs, either in clear text, or in JSON.

The 3G Bridge does not employ a garbage collection system, the client side—here, the modified computing element—has to explicitly delete a job after execution. After deleting the job, no information about it will be accessible—including the accounting information. Because of this, the accounting information must always be queried *before* the job is deleted.

In order to seamlessly integrate EGI accounting, the mCE functions should be extended. After a job has finished, before deleting it, its accounting metrics are acquired from the Bridge. This information is amended with required administrative information and is stored for the APEL client to access. In the mCE, the APEL client periodically checks the changes and synchronizes the database with the site APEL node, which regularly sends the accounting information to the central EGI accounting system.

6 Related work

There have been several technologies developed for gLite to support parametric study applications. The gLite WMS itself supports parametric study and collection jobs, but these techniques impose huge overhead on the gLite infrastructure, as the jobs in these collections have to be handled individually by the WMS.

The main components of this overhead are administrative tasks and finding available resources; impairing mostly the submission time of jobs. To reduce the submission overhead, *pilot* systems (like DIRAC [23] or Diane [24]) has been developed. In these systems, a single pilot job is submitted through gLite, as a placeholder, to a given CE. The pilot job *pulls* jobs from the pilot system's job repository for execution on the CE, and transfers information and results back. Pulling jobs and all communication are executed by-passing the gLite infrastructure, reducing the overhead of job submission from linear to constant time.

Pilot systems use the existing infrastructure to execute larger sets of arbitrary jobs. In contrast, the EDGI solution provides not only a middleware, but also computing resources to support service grid users. The Metajob feature enables users to execute parameter studies in the EDGI infrastructure with high efficiency. On the other hand, pilot systems can execute arbitrary jobs, while the desktop grid was constrained to a set of supported (ported and preregistered) applications. However, this disadvantage has been diminished with the introduction of GBAC [14], which enables users to submit arbitrary (non-ported, non-registered) applications to the desktop grids.

Generally, the pull method eliminates the need for active polling of the status of CEs, and therefore, is suitable for volatile environments and for handling large number of jobs. Because of this, the BOINC desktop grid itself has been designed as a pull-based system. The EDGI infrastructure with the metajob feature of the 3G Bridge enables gLite users to exploit the resources offered by the desktop grid, without the overhead of the push-based gLite system.

Condor [7] is a job-scheduler system that is able to handle parameter sweep type jobs. The syntax of the Metajob definition language was inspired by the job description language of Condor. In the condor submit file the user has the possibility to utilize existing macros (like “$(Process)”) to perform indexing in filenames. This feature is missing in our Metajob feature, but can be easily added in the future. Moreover, Condor can monitor the individual jobs themselves and can handle the results of the individual jobs as well, while Metajob cannot. Most of these missing features are coming from the restriction that Metajob must simulate the list of PS jobs as one single job.

Another interesting related work is the Condor-BOINC integration [25] which enables Condor to submit (parametric) jobs to BOINC. This solution integrates the power of Condor submission mechanism

ACCEPTED MANUSCRIPT

with the scalable job handling mechanism of BOINC. This solution is a really powerful mechanism; however, in our solution, the gLite, ARC and UNICORE as submission interfaces were already a constraint.

7 Conclusions

The Metajob concept introduced in this paper enables the submission of high number of jobs to desktop grids (BOINC as an example). This concept has the following very important advantages. (1) Simple and low-level parameter sweep description can be created very easily by the user or even by a simple script or a high-level tool. (2) Simple monitoring facility through a webpage where the URL can be easily propagated in service grid systems, like gLite. (3) Simple identification mechanism by returning the annotated submission file. (4) Solution is transparent from the point of view of the source service grid; therefore, it can also be utilized through ARC or Unicore. (5) The solution is transparent from the target infrastructure’s aspect as the Metajob concept is implemented by a separate 3G Bridge plugin. Therefore, other desktop grids, like XtremWeb or even new type of infrastructures can also be supported.

The concept is general enough to be integrated to any Grid middleware since the only extension is an additional input file (Metajob description) while the output is a compressed file of multiple results.

Among its advantages, the drawback of our solution is that it does not support standard job description languages; however, due to its simplicity, a converter for this purpose will also be implemented in the future.

The EDGI (and later its follow-up FP7 project, called IDGF-SP) maintains a service grid to desktop grid infrastructure where several BOINC and XtremWeb desktop grid sites collected more than 130 thousands of desktop and volunteer PCs worldwide. These sites and their resources can be accessed by gLite, ARC or Unicore users.

The technology reported in this paper makes the extension of gLite VOs with desktop grids a production level reality for every gLite VO and gLite user who would like to run large parameter study applications in a fast and efficient way.

Acknowledgement

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no 261556 (EDGI) and under grant agreement no 312297 (IDGF-SP).

References

[1] Cristina Aiftimiei, Paolo Andreetto, Sara Bertocco, Simone Dalla Fina, Alvise Dorigo, Eric Frizziero, Alessio Gianelle, Moreno Marzolla, Mirco Mazzucato, Massimo Sgaravatto, Sergio Traldi, Luigi Zangrando, Design and implementation of the gLite CREAM job management service, Future Generation Computer Systems, Volume 26, Issue 4, April 2010, Pages 654-667, ISSN 0167-739X, 10.1016/j.future.2009.12.006.
[2] A. Streit, P. Bala, A. Beck-Ratzka, K. Benedyczak et al., “Unicore 6 - recent and future advancements,” Berichte des Forschungszentrums Julich, vol. 65, 2010.
[3] Globus Toolkit Version 4: Software for Service-Oriented Systems. I. Foster. IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779, pp 2-13, 2006.
[4] M. Ellert, M. Grønager, A. Konstantinov, B. Konya, J. Lindemann, I. Livenson, J. Nielsen, M. Niinimaki, O. Smirnova, and A. Waananen, “Advanced resource connector middleware for lightweight computational grids,” Future Generation Computer Systems, vol. 23, no. 2, pp. 219 – 240, 2007.
[5] David P. Anderson. 2004. BOINC: A System for Public-Resource Computing and Storage. In Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing (GRID '04). IEEE Computer Society, Washington, DC, USA, 4-10. DOI=10.1109/GRID.2004.14 http://dx.doi.org/10.1109/GRID.2004.14

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

[6] G. Fedak, C. Germain, V. Neri, F. Cappello: XtremWeb: a generic global computing system, Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on In Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on (2001), pp. 582-587. doi:10.1109/CCGRID.2001.923246
[7] Douglas Thain, Todd Tannenbaum, and Miron Livny, "Distributed Computing in Practice: The Condor Experience" *Concurrency and Computation: Practice and Experience*, Vol. 17, No. 2-4, pages 323-356, February-April, 2005.
[8] The EDGI EU FP7 project: http://edgi-project.eu
[9] P. Kacsuk, J. Kovacs, Z. Farkas, A. Marosi, and Z. Balaton. Towards a powerful european DCI based on desktop grids. Journal of Grid Computing, 9:219-239, 2011.
[10] C. Ulrik Søttrup, Wäänänen, A., and Kovács, J., Transparent execution of ARC jobs on Desktop Grid resources, MIPRO, 2012, vol. Proceedings of the 35th International Convention. Opatija, pp. 271 - 276, 2012.
[11] Keller, Matthias and Kovács, József and Brinkmann, Andre (2011) Desktop Grids opening up to UNICORE. In: UNICORE summit 2011. Proceedings. Torun, 2011.
[12] Attila Csaba Marosi, Gabor Gombas, Zoltan Balaton, Peter Kacsuk, Enabling Java applications for BOINC with DC–API, in: Distributed and Parallel Systems, Proceedings of the 7th International Conference on Distributed and Parallel Systems, 2009, pp. 3–12.
[13] Attila Csaba Marosi, Zoltan Balaton, Peter Kacsuk, GenWrapper: a generic wrapper for running legacy applications on desktop grids, in: 3rd Workshop on Desktop Grids and Volunteer Computing Systems, PCGrid 2009, Rome, Italy, May, 2009.
[14] Attila Marosi, József Kovács, Peter Kacsuk, Towards a volunteer cloud system, Future Generation Computer Systems, Available online 27 March 2012, ISSN 0167-739X, 10.1016/j.future.2012.03.013.
[15] SpeQuloS: A QoS Service for BoT Applications Using Best Effort Distributed Computing Infrastructures Simon Delamare, Gilles Fedak, Derrick Kondo, Oleg Lodygensky. in International Symposium on High Performance Distributed Computing (HPDC'2012), Delft, Nederlands, 2012
[16] European Grid Infrastructure (EGI). http://www.egi.eu/, 2012
[17] Terstyanszky, Gabor and Kiss, Tamas and Kukla, Tamas and Lichtenberger, Zsolt and Winter, Stephen and Greenwell, Pamela and McEldowney, Sharron and Heindl, Hans (2012) Application repository and science gateway for running molecular docking and dynamics simulations. Healthgrid applications and technologies meet science gateways for life sciences. Studies in health technology and informatics (175). IOS Press, pp. 152-161. ISBN 9781614990536
[18] Z. Farkas, P. Kacsuk, Z. Balaton, G. Gombás, Interoperability of BOINC and EGEE, Future Generation Computer Systems, Volume 26, Issue 8, October 2010, Pages 1092-1103, ISSN 0167-739X, 10.1016/j.future.2010.05.009. (http://www.sciencedirect.com/science/article/pii/S0167739X10000890)
[19] EDGI project deliverable D6.1 "Integrated ARC, Desktop Grid, and Eucalyptus bridge", http://edgi-project.eu/documents/10515/50865/EDGID61.pdf
[20] European Middleware Initiative (EMI). http://www.eu-emi.eu/, 2012.
[21] Apel. https://wiki.egi.eu/wiki/APEL, 2012.
[22] Apel documentation, https://twiki.cern.ch/twiki/pub/EMI/APELClient/APEL-Messaging-v2.2.pdf, 2012
[23] A. Tsaregorodtsev et al., DIRAC: a community grid solution, J. Phys. Conf. Ser. 119 (2008) 062048.
[24] Vladimir V. Korkhov, Jakub T. Moscicki, and Valeria V. Krzhizhanovskaya. 2009. Dynamic workload balancing of parallel applications with user-level scheduling on the Grid. Future Gener. Comput. Syst. 25, 1 (January 2009), 28-34. DOI=10.1016/j.future.2008.07.001 http://dx.doi.org/10.1016/j.future.2008.07.001
[25] Documentation on Condor-BOINC integration, http://boinc.berkeley.edu/trac/wiki/CondorBoinc, 2013

ACCEPTED MANUSCRIPT

*Biographies (Text)

ACCEPTED MANUSCRIPT

József Kovács was born in 1975 in Budapest, Hungary. He is a Senior Research Fellow at the Laboratory of Parallel and Distributed Systems (LPDS) at the Computer and Automation Research Institute of the Hungarian Academy of Sciences. He got his B.Sc. (1997), M.Sc. (2001) and Ph.D. (2008). In the meanwhile he was continuously involved in numerous national, international and European (Esprit, FP5, FP6, FP7) research projects. From 2002 he was doing research on the field of parallel checkpointing techniques in grids and later on in the field of DesktopGrid computing. From 2006 he is the leader of the DesktopGrid team of LPDS. He is the author and co-author of more than 40 scientific papers.

ACCEPTED MANUSCRIPT

*Biographies (Text)

ACCEPTED MANUSCRIPT

Ádám Visegrádi is a member of the Laboratory of Parallel and Distributed Systems at the Computer and Automation Research Institute of the Hungarian Academy of Sciences since 2009. He received his M.Sc. in computer science from the Eötvös Loránd University and started his Ph.D. studies there in 2012. His research interests include desktop grid and cloud computing and, particularly, their data-intensive applications. He participated in national (Web2Grid) and international (EDGI) research and development projects.

ACCEPTED MANUSCRIPT

*Biographies (Text)

ACCEPTED MANUSCRIPT

Peter KACSUK is the Director of the Laboratory of the Parallel and Distributed Systems in the Computer and Automation Research Institute of the Hungarian Academy of Sciences. He received his MSc and university doctorate degrees from the Technical University of Budapest in 1976 and 1984, respectively. He received the kandidat degree (equivalent to PhD) from the Hungarian Academy in 1989. He habilitated at the University of Vienna in 1997. He recieved his professor title from the Hungarian President in 1999 and the Doctor of Academy degree (DSc) from the Hungarian Academy of Sciences in 2001. He served as full professor at the University of Miskolc and at the Eötvös Lóránd University of Science Budapest. He has been a part-time full professor at the Cavendish School of Computer Science of the University of Westminster. He has published two books, two lecture notes and more than 200 scientific papers on parallel computer architectures, parallel software engineering and Grid computing. He is co-editor-in-chief of the Journal of Grid Computing published by Springer.

ACCEPTED MANUSCRIPT

*Biographies (Photograph)
Click here to download high resolution image
ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

*Biographies (Photograph)
Click here to download high resolution image

ACCEPTED MANUSCRIPT

*Biographies (Photograph)

Click here to download high resolution image