Design and Implementation of Condor-UNICORE Bridge

Hidemoto Nakada
National Institute of Advanced Industrial
Science and Technology (AIST)
1-1-1 Umezono, Tsukuba, 305-8568, Japan
hide-nakada@aist.go.jp

Jaime Frey
University of Wisconsin
1210 W Dayton St, Madison, WI 53706, U.S.
jfrey@cs.wisc.edu

Motohiro Yamada, Yasuyoshi Itou
Fujitsu Limited
1-9-3, Nakase, Chiba City,
Chiba, 261-8588, Japan
moto.yamada@jp.fujitsu.com, itou@strad.ssg.fujitsu.com

Yasumasa Nakano
Fujitsu Limited
140 Miyamoto, Numazu,
Shizuoka, 410-0396 Japan
nakano@soft.fujitsu.com

Satoshi Matsuoka
Tokyo Institute of Technology,
2-12-1 Ookayama, Tokyo, 152-8550, Japan
matsu@is.titech.ac.jp

Abstract

In this paper, we describe the design and implementation of a generic grid interface for Condor. Though Condor has intefaces for specific grid systems, such as Globus Toolkit 2 GRAM, it is not easy to add new interfaces for other grid systems, since it requires some code modification inside the Condor. With our new interface design, supporting a new grid system can be established without any code modification in the Condor core itself. We also implemented a bridge for the UNICORE system and validated that our approach is effective.

1. Introduction

Condor is a job queuing system, developed at the University of Wisconsin, aiming to achieve high-throughput computing utilizing unused computers within the campus. It also can work as a meta-scheduler that harnesses remote schedulers via grid-aware remote execution systems, such as Globus GRAM [3]. Using Condor, users can submit their jobs to remote sites through the grid-aware systems. Although Condor has an interface mechanism to communicate with grid-aware systems, the interface mechanism is not well generalized. Adding a new interface to a new grid-aware system requires code modification inside the Condor core.

We designed a new interface architecture for Condor to talk with external grid-aware systems that enables the addition of new external system support without modifying the Condor core. Based on the architecture, we implemented a new bridge that connects Condor with UNICORE[5, 2], a grid-aware system, to validate the approach.

The rest of the paper is structured as follows. In Sections 2 and 3, we give a brief description of Condor and UNICORE, respectively. In Section 4, we propose the design of the generic interface architecture. In Section 5, we give a detailed explanation of the implementation of the bridge to UNICORE. Section 6 concludes the paper, giving a summary and our future work.

2. Overview of Condor

Condor is a job queuing system, developed at the University of Wisconsin, aiming to achieve high-throughput computing utilizing unused computers within the campus. Recently, it has also been used as a meta-scheduler for Globus [3]-managed resources. Condor manages numerous computer resources and assigns them to submitted jobs from users.

Condor uses a flexible mechanism called

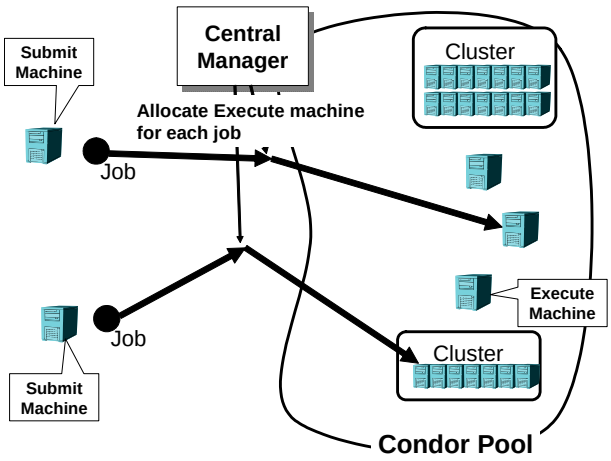

Figure 1. Overview of Condor.

'matchmaking'[4] to allocate the resources for the jobs. Condor also supports user program checkpointing and preemption to enable flexible priority based job scheduling [7].

2.1. Condor Architecture

Figure 1 shows an overview of the Condor System. Computers participating in a Condor pool can have three roles: central manager, submission machine, and execution machine. A computer can play multiple roles. A Condor pool has to have exactly one central manager and one or more submission machines and one or more execution machine. Users submit jobs from the submission machine, and the central manager takes care of allocating an execution machine for each and every job.

A submission machine embodies a job queue and sends job information periodically to the central manager. An execution machine monitors its own resource information, such as load average and amount of free memory, and reports it to the central manager periodically. The central manager takes this information and decides on a suitable execution machine to be matched with a queued job.

Figure 2 shows the Condor daemons. On the submit machine, there is a daemon called the Schedd (Scheduler Daemon), that manages a job queue. Each Schedd has its own independent job queue. Users submit jobs to the Schedd and the Schedd sends the job information to the Collector on the central manager. On the execute machine, a daemon called the Startd (Start Daemon) monitors the status of the machine and reports it to the Collector periodically.

On the central manager, the Negotiator matches jobs and

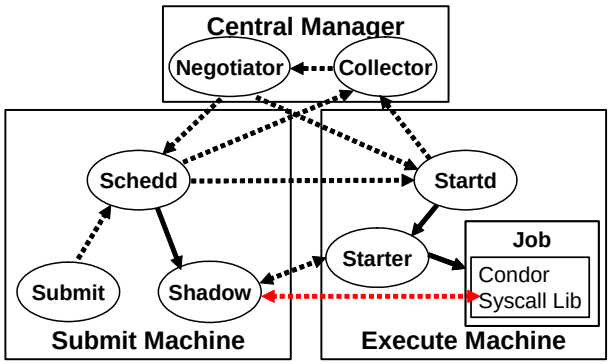

Figure 2. Condor daemons.

resources using the information gathered by the Collector. When a job is matched to a resource, the Negotiator informs the Schedd and Startd. The Schedd sends the job to the Startd for execution.

2.2. ClassAds

ClassAds are the source of flexibility in Condor. They are modeled after the classified ads in the newspaper. A ClassAd describes the attributes of an entity and the preferences and requirements that entity is looking for in a match. A match-maker makes the best matches between entities within the requirements given. In Condor, the entities are jobs and resources and the Negotiator performs the matchmaking. ClassAds are schema-less, which makes it easy for users to add new attributes.

2.3. Condor-G

Condor can submit jobs to compute resources that are managed by GRAM, the resource manager provided as a part of the Globus Toolkit. This facility is called Condor-G[6].

Figure 3 shows the diagram for job invocation in Condor-G. In Condor-G, a Globus GRAM server replaces the Condor Startd as the manager of the execution resource. Instead of a Shadow, the Schedd starts a GridManager, which talks to the Globus GRAM server. The GridManager invokes another process called the Globus GAHP server that is responsible for Globus protocol handling. The GridManager and Globus GAHP server talk each other with a protocol called GAHP[1], which will be described below.

After being authenticated by the GRAM gatekeeper, the GAHP server submits a job request to the jobmanager. The jobmanager then submits the job to the local batch queueing system, which finally executes the job.

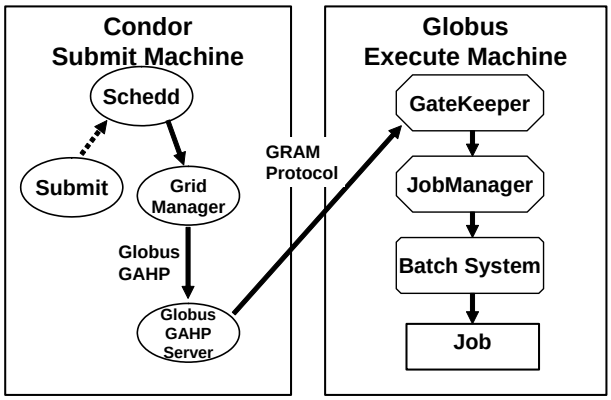

Figure 3. Job Submission by Condor-G.

2.4. GAHP Overview

GAHP (Grid ASCII Helper Protocol) is a text-based protocol designed to let the GridManager use grid software that it cannot be easily linked with. The library is isolated in a seperate process (called the GAHP server) that the GridManager communicates with via a pair of pipes. The Gridmanager issues requests to invoke operations in the grid software and the GAHP server replies with the results of those operations. The protocol is asynchronous and non-blocking with respect to the requested operations. The GridManager can have many requests outstanding, the operations executing in parallel in the GAHP server.

Figure 4 shows the exchange for one operation. The client (the GridManager) sends a **Request Line** to the server, consisting of the operation name, an identifier for the request, and a list of input parameters. The server replies immediately with a **Return Line** indicating the request is understood and the operation will be carried out. The client is then free to perform other tasks, including issuing more requests. Once the operation completes, the server sends a **Result Line**, which includes the request id, whether the operation was successful, and a list of output parameters.

Each line is terminated by **CR LF** and the fields within the line are seperated by spaces. Spaces, **CR LF**s, and backslashes within the fields must be escaped with a backslash.

The base GAHP protocol defines a framework for communication. To use it, a specific command set has to be defined. This includes command names and what parameters they use. The Globus GAHP server uses its own command set specific to Globus GRAM.

3. UNICORE Overview

UNICORE is a grid-aware middleware developed mainly by Fujitsu Laboratory Europe. The main goal of the

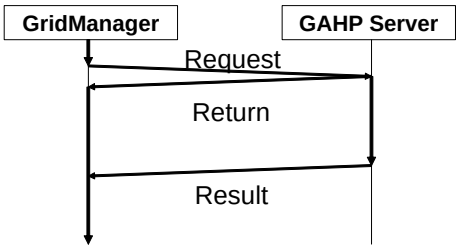

Figure 4. GAHP message communication.

system is to enable the seamless utilization of supercomputers residing in several geographically distributed supercomputer centers. The goal required UNICORE to have firewall-aware architecture.

One of the paramount features of UNICORE is workflow management. UNICORE has a workflow engine embedded in it, and users can submit a group of jobs specifying dependencies among the jobs. The job workflow is represented as a Java object, called AJO (Abstract Job Object).

Figure 5 shows an overview of UNICORE. UNICORE has a two-level abstraction for sites. One level is called Usite (Unicore Site), that stands for organizations surrounded by firewalls, such as a supercomputer center. Each Usite has a daemon called Gateway, that runs on the firewall and relay all the communication to/from the Usite.

Each Usite has one or more Vsites (Virtual Sites) in it, which stands for a scheduler queue for computation. A Vsite can be a single computer, a cluster that shares a scheduling queue, or a supercomputer. Each Usite has a daemon called NJS (Network Job Supervisor), that is responsible for job execution. NJS also works as a workflow engine, cooperating with another daemon called TSI.

TSI(Target System Interface) is a wrapper script that abstracts local job schedulers such as PBS or LSF.

4. Design of a Generic Interface for Grid-Aware Systems

4.1. Condor-G GAHP protocol

As mensioned in section 2.3, Condor-G has the GridManager and Globus GAHP server as separate modules, and all the code that uses Globus API is inside the GAHP server.

Readers might think that it would be easy to rewrite the GAHP server to support a new grid-aware remote execution system, but that is not the case. The problem is that the logic to control the Globus resides in the GridManager, not in the Globus GAHP server. As the result, the Globus GAHP

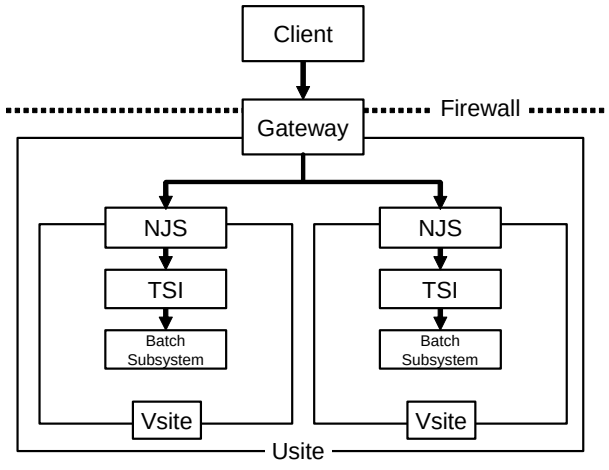

Figure 5. Overview of UNICORE.

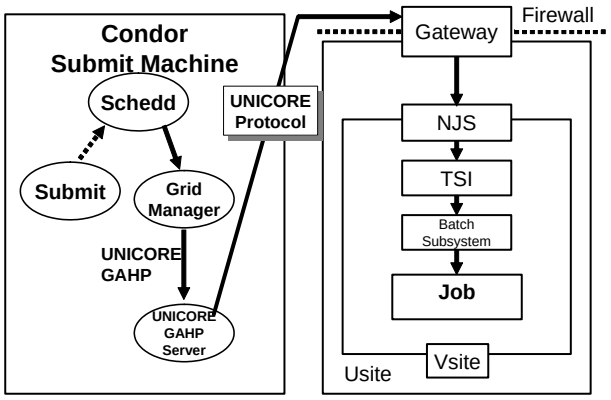

Figure 6. Job Submission using the UNICORE bridge.

server command set has Globus-specific, complicated semantics. The Globus GAHP server is just a stub program to communicate with the Globus modules, and all the program logic is in the GridManager.

With this approach, supporting a new grid aware system would require 1) rewriting the GridManager to implement the system specific job control logic, 2) defining a GAHP command set that is specific for the system, and 3) writing a GAHP server that supports the command set.

The hardest part is the first one, since the GridManager is a very complicated program that includes Condor specific implementation details.

4.2. Generic GAHP command set

To cope with the issue discussed above, we defined a generic GAHP command set that can handle any grid-aware system. The command set is designed to be grid system neutral. With this command set, supporting a new grid-aware system can be achieved just by re-implementing the GAHP server.

Table 1 shows the generic GAHP command set. <S|F> means that the string will be 'S' or 'F', which stand for 'Success' and 'Failure'. <S|E> means that the string will be 'S' or 'E', which stands for 'Success' and 'Error'.

Note that the command set itself does not have any specific features for UNICORE, even though the command name includes 'UNICORE'.

The generic command set is composed of 5 commands: create, start, status, recover, and destroy. We used the ClassAd format to represent job requirements and job status. The reasons are 1) ClassAds are used everywhere in the Condor implementation and can be easily handled with in Condor, and 2) ClassAds are flexible, allowing designers to add arbitrary attributes.

ClassAds have two text notation forms. One is XML based and the other is a propretary format. We used the XML based notation.

5. Implementation of the UNICORE Bridge with Generic GAHP Command Set

To implement a bridge to a grid-aware system using the generic GAHP command set, we have to define a system specific ClassAd attribute set.

Since the GAHP protocol is based on simple text format, the GAHP server can be implemented in any language. We choose Java as the implementation language because the UNICORE protocol is strongly tied to the Java implementation.

Figure 6 shows job execution steps with the UNICORE bridge.

5.1. Defining ClassAd Attributes

The ClassAd transferred between the GridManager and the UNICORE GAHP server has to contain enough information to create jobs and represent job status. Table 2 shows attributes used in the UNICORE GAHP server. The first set are also also in the other portions of the Condor system, and the second set are only used for UNICORE GAHP.

The UNICORE GAHP server returns Job Status to the Condor GridManager. The problem here is that there is a semantic gap between Condor and UNICORE job status. While in UNICORE job status is represented by three layered complex values, in CONDOR there are just 6 values to

NAME	JOB_CREATE
Request	`UNICORE_JOB_CREATE <req id> <job classad>`
Return	`<S|E>`
Result	`<req id> <S|F> <job handle> <error string>`
NOTE	Creates a job and returns the handle to the job. Description of the job is provided as a ClassAd.
NAME	JOB_START
Request	`UNICORE_JOB_START <req id> <job handle>`
Return	`<S|E>`
Result	`<req id> <S|F> <error string>`
NOTE	Starts the job specified by the job handle.
NAME	JOB_STATUS
Request	`UNICORE_JOB_STATUS <req id> <job handle>`
Return	`<S|E>`
Result	`<req id> <S|F> <result classad> <error string>`
NOTE	Returns the status of the job specified by the job handle. Status is represented as the ClassAd.
NAME	JOB_RECOVER
Request	`UNICORE_JOB_RECOVER <req id> <job classAd>`
Return	`<S|E>`
Result	`<req id> <S|F> <error string>`
NOTE	Recovers a job that is already submitted. This command is used when the GAHP server is restard by GridManager.
NAME	JOB_DESTROY
Request	`UNICORE_JOB_DESTROY <req id> <job handle>`
Return	`<S|E>`
Result	`<req id> <S|F> <error string>`
NOTE	Destroys the job specified by the job handle. If the job is still alive, the GAHP server will cancel it first, and then release the information stored inside the GAHP server. It also tries to release information stored inside the target grid system, Unicore.

represent it. The UNICORE GAHP server maps the UNICORE job status into Condor job status according to the map shown in table 3, and returns it to the GridManager.

5.2. Pass phrase management

In UNICORE, clients and servers authenticate each other with X.509 certificates. The UNICORE GAHP server acts as a client in the UNICORE world, hence it needs to be able to access the certificate, that is stored in the Java standard KeyStore file. To access it, the GAHP server has to know the pass phrase for it.

Readers might think that it will be accomplished by prompting users on startup, but it is not acceptable since the GAHP server will be rebooted automatically when the client machine crashes. Users are not on the console at all times.

We implemented the GAHP server so that it assumes the pass phrase is stored in a file that will be only readable by the user's account. The GAHP server opens and reads the file to get the passphrase and, using the passphrase, opens the keystore and retrieves the user's certificate.

5.3. Implementation of UNICORE GAHP server

The UNICORE GAHP Server shares its core part with the GAHP server for Globus Toolkit 4. They both are implemented in Java and share a package that is capable of handling the basic GAHP protocol. The UNICORE specific portion is implemented in different package.

The handler for each command has to be implemented as a class. The mapping from command to class name is supplied as a properties file. Hence, implementing a new GAHP server does not require any modification of the core code. All the programmer has to do is implement classes for each command and write a map file for them.

5.4. UNICORE Job Submission from Condor

5.4.1. Writing the Submission File To submit a job to UNICORE system from Condor, users have to write their

Attribute		Ex.
Cmd	executable file path	/home/foo/a.exe
Args	arguments given to the executable	arg1, -a
Env	environment variables for the execution	LANG=en_US
IWD	client side base directory	/home/nakada/condor
In	input file to be read in from stdin	input.dat
Out	filename to store output from stdout	ouput.dat
Err	filename to store output from stderr	Error.dat
TransferInput	filenames to be staged to the execute machine	a.exe, input.dat
TransferOutput	filenames to be staged back to the client	out.dat
JobStatus	Condor job status	refer to the table 3 second column
ErrMessage	Detailed Error massage	Script reported no errors
RemoteWallClockTime	Job execution time duration	123.0
ByteSent	number of bytes that are sent	1023004
ByteRecvd	number of bytes that are received	1023004
ExitBySignal	Is the process killed by signal?	TRUE
ExitCode	Exit code of the process	1
ExitSignal	Signal number causing the death of the process	9
Upper Condor Generic / Lower for UNICORE only		
UnicoreUsite	Unicore Usite gateway FQDN and port number	fujitsu.com:1234
UnicoreVsite	Vsite name in the usite	NaReGI
KeystoreFile	keystore filename	/home/foo/key
PassphraseFile	passphrase filename	/home/foo/passwd
UnicoreJobId	handle job id	fujitsu.com:1234/NaReGI/1374036929
UnicoreJobStatus	Unicore job status	refer to table 3 third column
UnicoreLog	UNICORE log filename	/var/log/unicore.log

Condor Job status	code	UNICORE Job status
I (Idle)	1	CONSIGNED PENDING READY EXECUTING QUEUED SUSPENDED
R (Running)	2	RUNNING
X (Removed)	3	
C (Completed)	4	SUCCESSFUL FAILED_IN_EXECUTION KILLED FAILED_IN_INCARNATION FAILED_IN_CONSIGN NEVER_TAKEN
H (Held)	5	HELD

Table 3. Job status mapping.

own submission file, just like to submit an ordinary Condor job. The only difference is that the users have to add few UNICORE specific attributes in the file.

The Condor submission file is similar to the ClassAd, but the syntax and attribute names are slightly different due to historical reasons.

UNICORE specific ClassAd attributes are not recognized by the Condor job submission tool, and will be filtered out automatically. To avoid the filter, users have to put a heading '+' to the non standard attribute names. These syntactical oddities will be cleaned up in the publically-released version. Figure 7 shows a sample submission file.

The first line '`universe = grid`' declares the job will be handled by an external grid-aware system, and the second line specifies that the system is UNICORE.

5.4.2. Job Submission To submit a UNICORE job, users use the `condor_submit` command, that is standard job submission command for Condor. Note that users have to prepare their keystore and passphrase file before making a submission.

The Schedd receives the submission file and translates it into ClassAd format, invokes the GridManager, and the GridManager invokes the UNICORE GAHP server. The GridManager creates a job by send-

```
⟶ REQUEST   UNICORE_JOB_CREATE 2 <c><a\ n="ClusterId"><i>1680</i></a><a\ n="QDate"><i>1122041597</i>
            </a><a\ n="CompletionDate"><i>0</i></a><a\ n="Owner"><s>nakada</s></a><a\ n="JobUniverse">
            <i>9</i></a><a\ n="JobGridType"><s>unicore</s></a><a\ n="TransferExecutable"><b\ v="f"/></
            a><a\ n="Cmd"><s>/bin/cat</s></a><a\ n="User"><s>nakada@a02.aist.go.jp</s></a><a\ n="Env">
            <s></s></a><a\ n="JobNotification"><i>2</i></a><a\ n="UserLog"><s>/usr/users/nakada/work/u
            nicoreGAHP/log</s></a><a\ n="In"><s>the_file</s></a><a\ n="StreamIn"><b\ v="f"/></a><a\ n
            ="Out"><s>out.1680.0</s></a><a\ n="StreamOut"><b\ v="f"/></a><a\ n="Err"><s>err.1680.0</s>
            </a><a\ n="StreamErr"><b\ v="f"/></a><a\ n="TransferFiles"><s>ONEXIT</s></a><a\ n="Transfe
            rInput"><s>the_file</s></a><a\ n="UnicoreUSite"><s>ghost:4433</s></a><a\ n="UnicoreVSite"
            ><s>ghost</s></a><a\ n="UnicoreKeystoreFile"><s>keystore2</s></a><a\ n="UnicorePassphraseF
            ile"><s>passphrase</s></a><a\ n="UnicoreUserAlias"><s>hidemoto\ nakada's\ unicore\ forum\
            e.v.\ id\ (1)</s></a><a\ n="Args"><s>the_file</s></a></c>
⟵ REPLY     S
⟵ RESULT    2 S 1798031550::1680.0 null

⟶ REQUEST   UNICORE_JOB_START 3 1798031550::1680.0
⟵ REPLY     S
⟵ RESULT    3 S null

⟶ REQUEST   UNICORE_JOB_STATUS 4 1798031550::1680.0
⟵ REPLY     S
⟵ RESULT    4 S <c><a n="UnicoreJobId"><s>1798031550</s></a></c> null

⟶ REQUEST   UNICORE_JOB_STATUS 5 1798031550::1680.0
⟵ REPLY     S
⟵ RESULT    5 S <c><a n="UnicoreJobId"><s>1798031550</s></a><a\ n="UnicoreJobStatus"><s>SUCCESSFUL</s>
            </a><a\ n="JobStatus"><i>4</i></a><a\ n="ByteSend"><r>3.200000000000000E+01</r></a><a\ n="
            ByteRecvd"><r>3.180000000000000E+02</r></a><a\ n="RemoteWallClockTime"><r>1.000000000000000
            0E+00</r></a><a\ n="ExitCode"><i>0</i></a><a\ n="ExitSignal"><i>0</i></a><a\ n="ExitBySign
            al"><b\ v="f"/></a></c> null

⟶ REQUEST   UNICORE_JOB_DESTROY 6 1798031550::1680.0
⟵ REPLY     S
⟵ RESULT    6 S null
```

Figure 8. GridManager - UNICORE GAHP Server Communication.

```
Universe   = grid
grid_type  = unicore

Executable = a.out
output     = tmpOut
error      = tmpErr
log        = tmp.log

+UnicoreUsite = fujitsu.com:1234
+UnicoreVsite = NaReGI
+KeystoreFile  = /home/foo/key

+PassphraseFile = /home/foo/passwd
Queue
```

Figure 7. A sample submit file.

ing the UNICORE_JOB_CREATE command with the ClassAd made from the submission file to the GAHP server.

The UNICORE GAHP server parses the ClassAd which is given as a argument of the create command and gets the name of the keystore file and the passphrase file. It read the contents and gets the user's certificate. It will use the certificate to communicate with the UNICORE servers.

Then, the UNICORE GAHP server creates an AJO from the ClassAd that describes the job requirements, given by the GridManager. Each AJO is assigned a unique AJO ID that is randomly created by the client library. Using the AJO ID as a part, the GAHP server composes a Job ID and returns it to the GridManager.

After receiving the Job ID and saving it to persistent storage, the GridManager will request to start the job with the UNICORE_JOB_START command with the Job ID. The GAHP server will submit the corresponding AJO to the UNICORE server.

5.4.3. Job Management The UNICORE GAHP server periodically checks the status of the job. When it finds a job is already done, it retrieves the output file from the server.

The GridManager also periodically polls job status to the GAHP server using `UNICORE_JOB_STATUS` command. When it finds that a job is done, it informs the Schedd. And then, it will destroy the job by issuing the `UNICORE_JOB_DESTROY` command to the GAHP server to release all the data and memory in the GAHP server and UNICORE servers.

Figure 8 shows an example of the communication sequence between the GridManager and the UNICORE GAHP server.

5.4.4. Recovery from Client Crash Condor can handle the remote jobs even when the submission machine crashes. The Schedd will be automatically rebooted and will reconnect to the remote servers.

The Schedd stores all the information on jobs it handles in a database. When it is rebooted, it scans the database, finds it has remote jobs and get the Job IDs for the jobs. It will reinvoke the GridManager and the GridManager will invoke the GAHP server, in turn. After the re-invocation, the GridManager will issue the recover command for each job, specifying the ClassAd for the job as a argument. In the ClassAd, JobID that was used in the last incarnation is stored, and in the JobID, a UNICORE AJO id is encoded that is necessary to reconnect the UNICORE job.

The GAHP server gets the AJO id form the JobID and restarts monitoring jobs in the UNICORE system.

6. Conclusion

We proposed the design of a generic interface to external grid systems for Condor, and proved the validity of the approach implementing a bridge to the UNICORE system based on the design.

The following is our future work:

- Reconsider passphrase storage
  In the current implementation, we store keystore passphrases into files. This might cause security issues. We have to find a better way that still allows the GAHP server to recover from failures without user interaction.

- Test the generality of the commnad set
  We designed the GAHP command set so that it can be used for any other Grid systems. We have to test the generality by actually applying the command set to some other grid system.

Acknowledgement

A part of this research was supported by a grant from the Ministry of Education, Sports, Culture, Science, and Technology (MEXT) of Japan through the NAREGI (National Research Grid Initiative) Project.

References

[1] Globus ASCII Helper Protocol. http://www.cs.wisc.edu/condor/gahp/.

[2] Unicore. http://www.unicore.org/.

[3] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. In *Proc. of Workshop on Environments and Tools, SIAM.*, 1996.

[4] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed resource management for high throughput computing. In *Proc. of HPDC-7*, 1998.

[5] M. Romberg. The unicore architecture - seamless access to distributed resources. In *Proc. of 8th IEEE International Symposium on High Performance Distributed Computing (HPDC8)*, pages 287–293, 1999.

[6] D. Thain, T. Tannenbaum, and M. Livny. Condor and the grid. In F. Berman, G. Fox, and T. Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons Inc., December 2002.

[7] T. Tannenbaum and M. Litzkow. Checkpointing and Migration of UNIX Processes in the Condor Distributed Processing System. In *Dr Dobbs Journal*, Februrary 1995.