

THE UNIVERSITY of EDINBURGH



Beyond the SAGA

From Simple APIs to Dynamic Abstractions

R.





- I. A Simple API for Grid Applications
- II. Distributed Programming Abstractions (DPA)
- III. A Good Research Idea ?
- IV. A Somewhat Concrete Plan



- Simple API for Grid (Distributed) Application
- Open Grid Forum community standard (GFD-R-P.90)
 - Describes a language-independent (SIDL), object oriented API for high-level tasks *considered useful* in distributed applications, like job submission, file transfer, communication, etc...
- Implementations of the GFD-R-P.90 standard:
 - JSAGA (Centre de Calcul IN2P3/CNRS, Lyon, France)
 - JavaSAGA (Vrije Universiteit, Amsterdam, NL)
 - C++ / Python Implementation (LSU, Baton Rouge, USA)

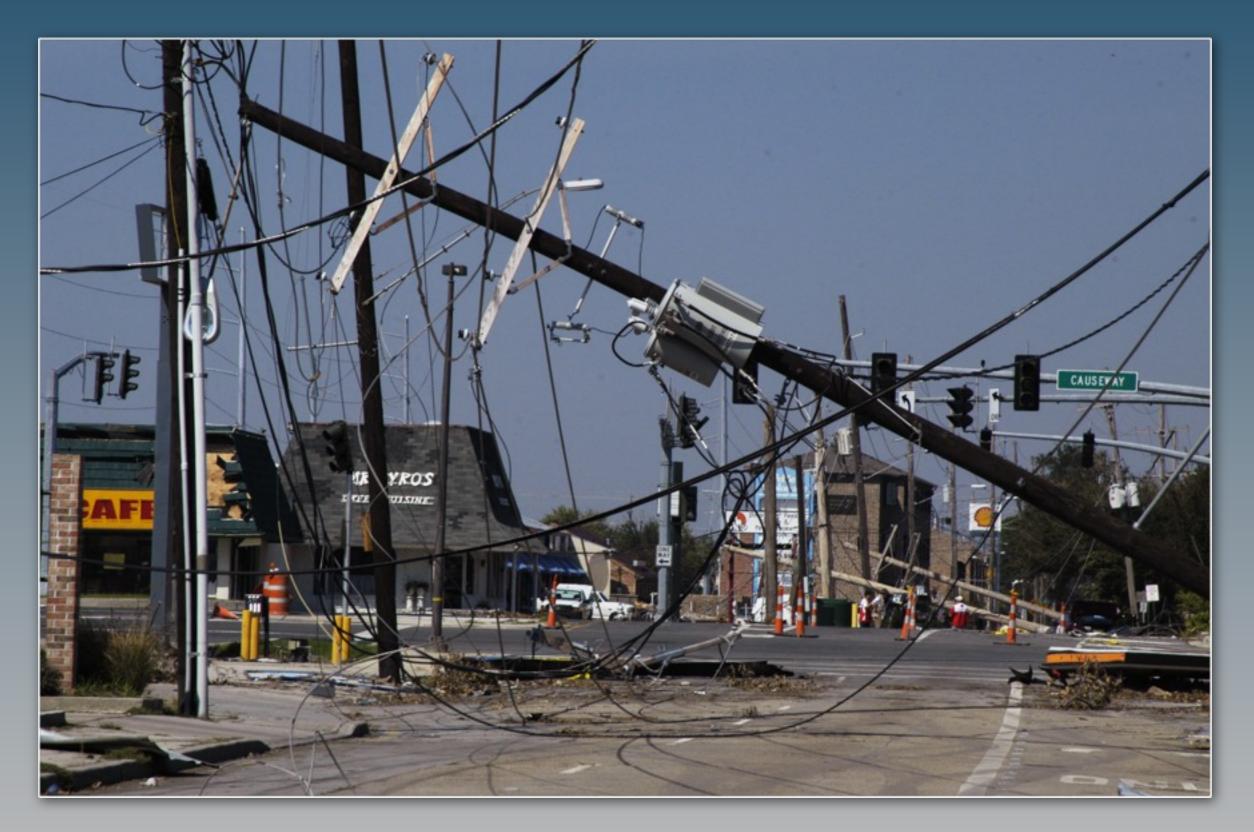


- 1988: Start of the Condor project
- 1998: Globus 1.0 released
- 2002: GridLab's Grid Application Toolkit (GAT)
- 2004: OGF (GGF) SAGA working group formed
- 2006: SAGA C++ development of the reference implementation starts at Louisiana State University
- 2010: GFD-90 1.0 standard (SAGA) released by OGF
- 2011: SAGA is being deployed across US TeraGrid, deployment on EGI, FutureGrid and XD is under review











- 1988: Start of the Condor project
- 1998: Globus 1.0 released
- 2002: GridLab's Grid Application Toolkit (GAT)
- 2004: OGF (GGF) SAGA working group formed
- 2006: SAGA C++ development of the reference implementation starts at Louisiana State University
- 2010: GFD-90 1.0 standard (SAGA) released by OGF
- 2011: SAGA is being deployed across US TeraGrid, deployment on EGI, FutureGrid and XD is under review

Why a Middleware-Independent API ?



- (Lifetime of applications) > (Lifetime of infrastructure and interfaces)
- Portability / adaptability = protection of assets
- Opens new opportunities for large-scale distributed systems research and evaluation
- Newly emerging extreme-scale simulations may have to span (scale-out) across several different infrastructures
- Distributed computing's counterpart of MPI ?

LSU's C++ / Python SAGA



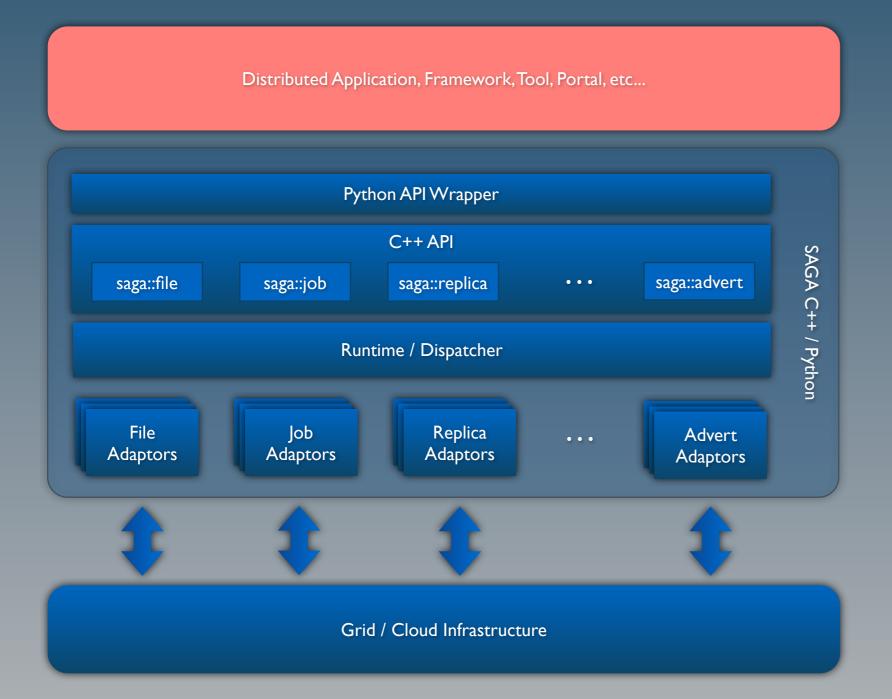
- Designed after the adaptor pattern
- A set of C++ libraries and headers grouped into functional packages
- Adaptors (plug-ins) that provide access to distributed middleware (Globus, gLite, Condor, etc...)
- A light-weight runtime/dispatcher that the right adaptor for an API call at runtime
- No services, daemons, etc...



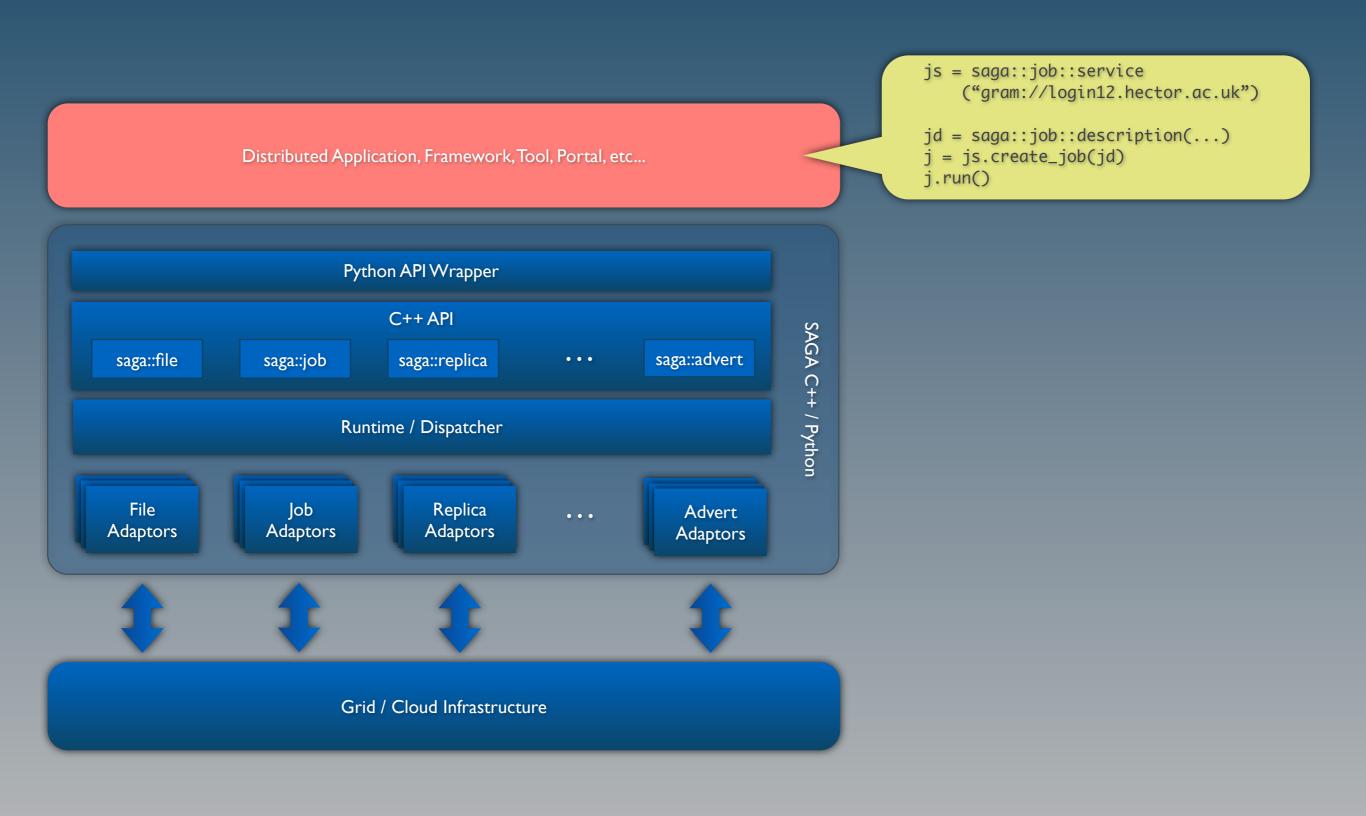




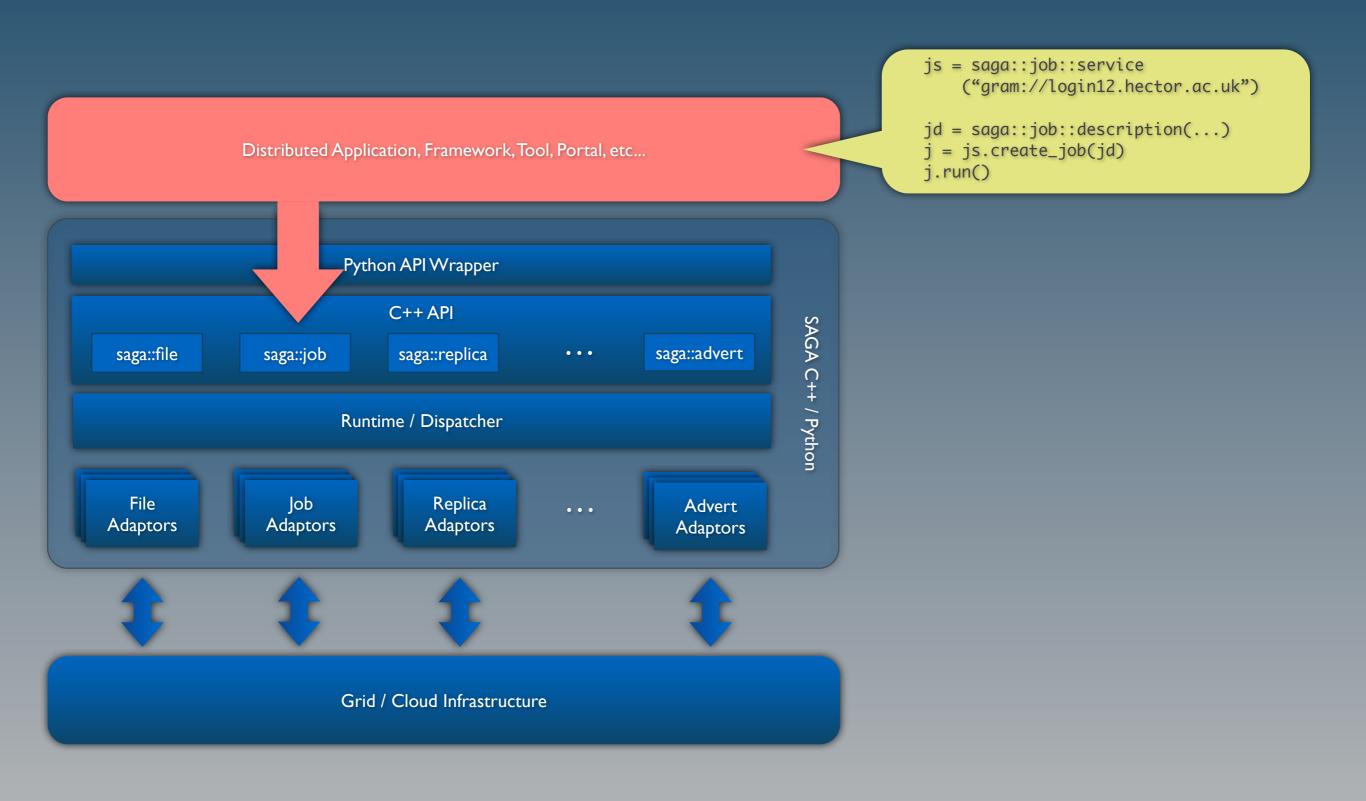






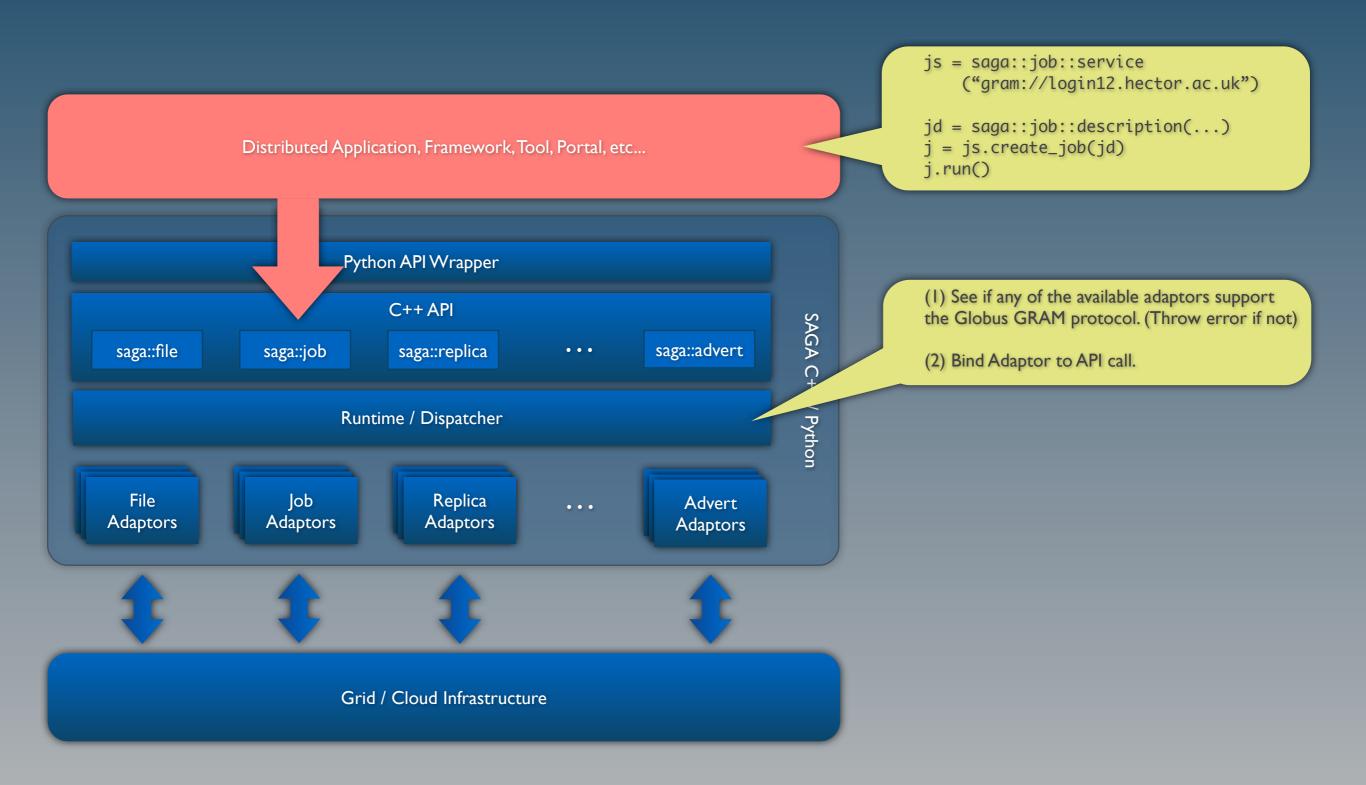




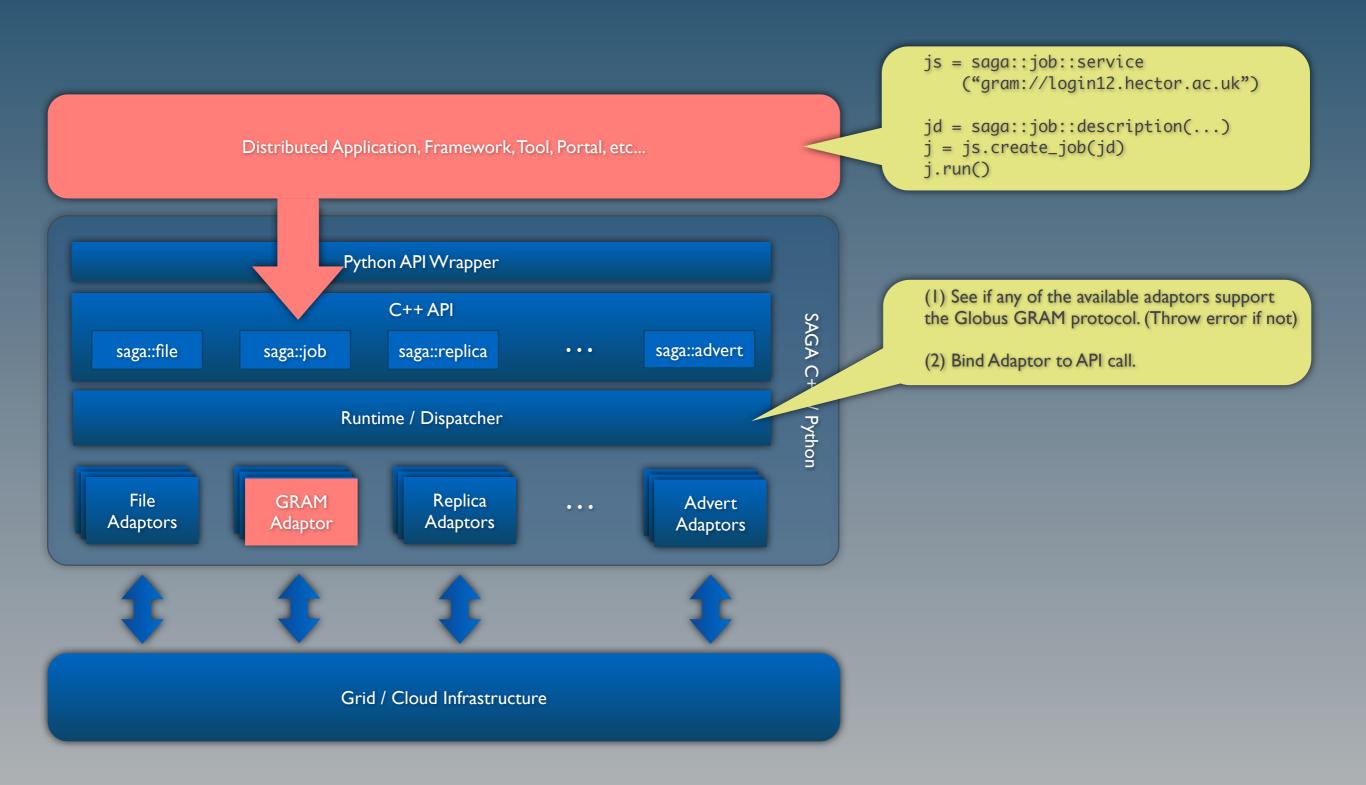




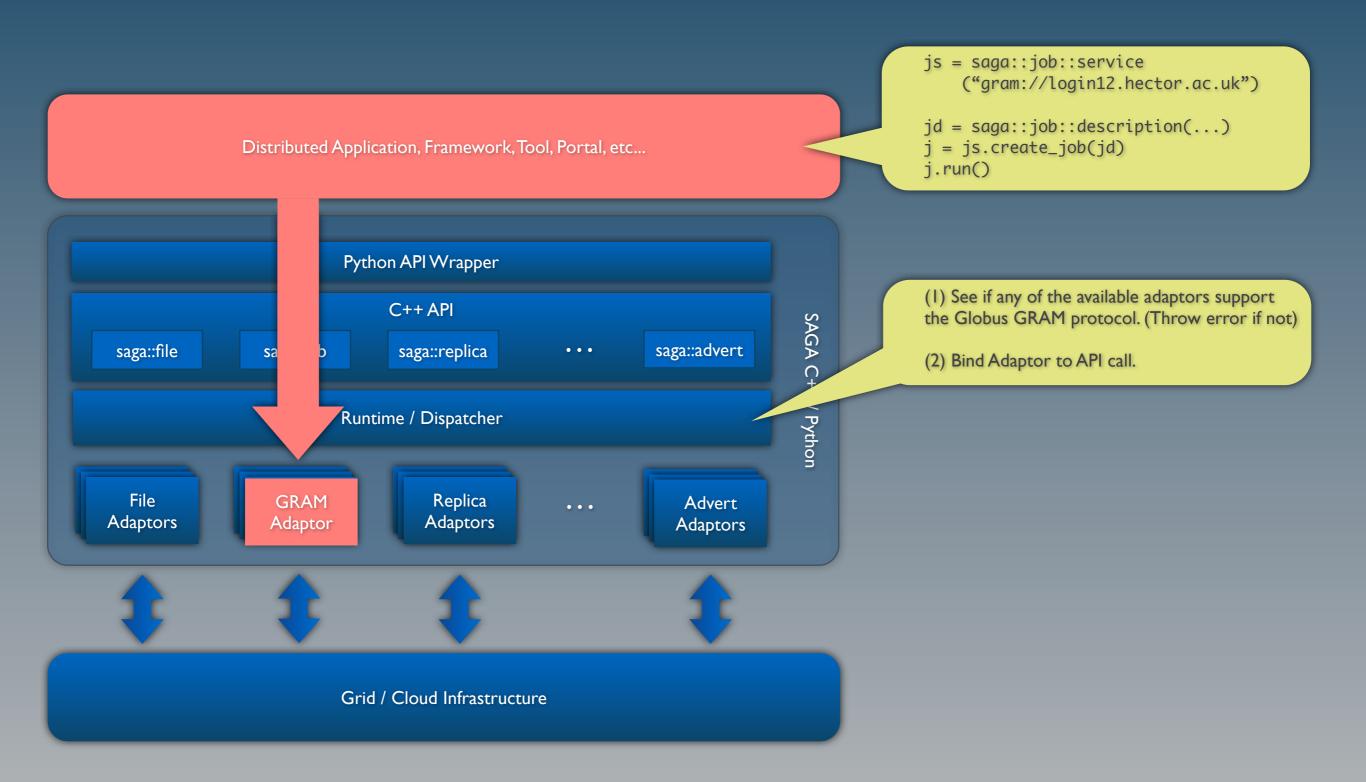
Mar 18, 2011



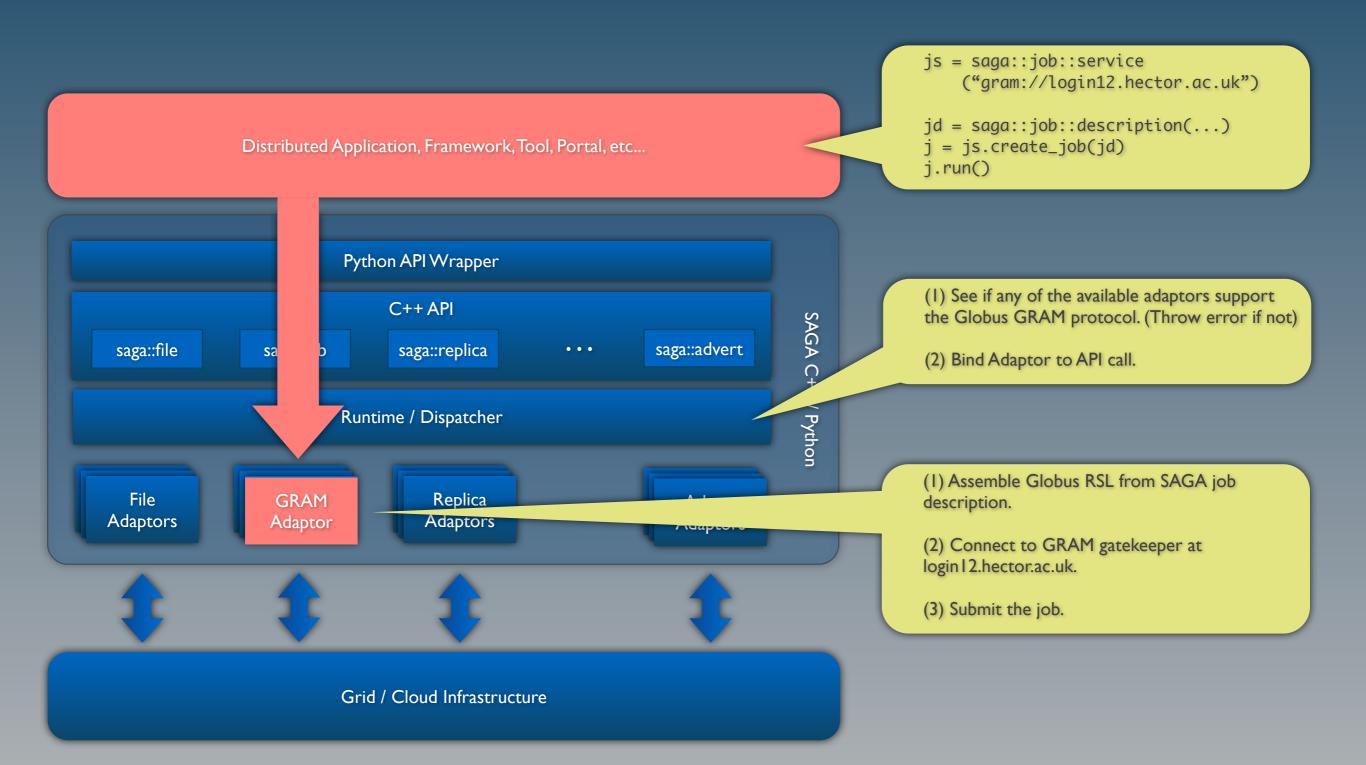




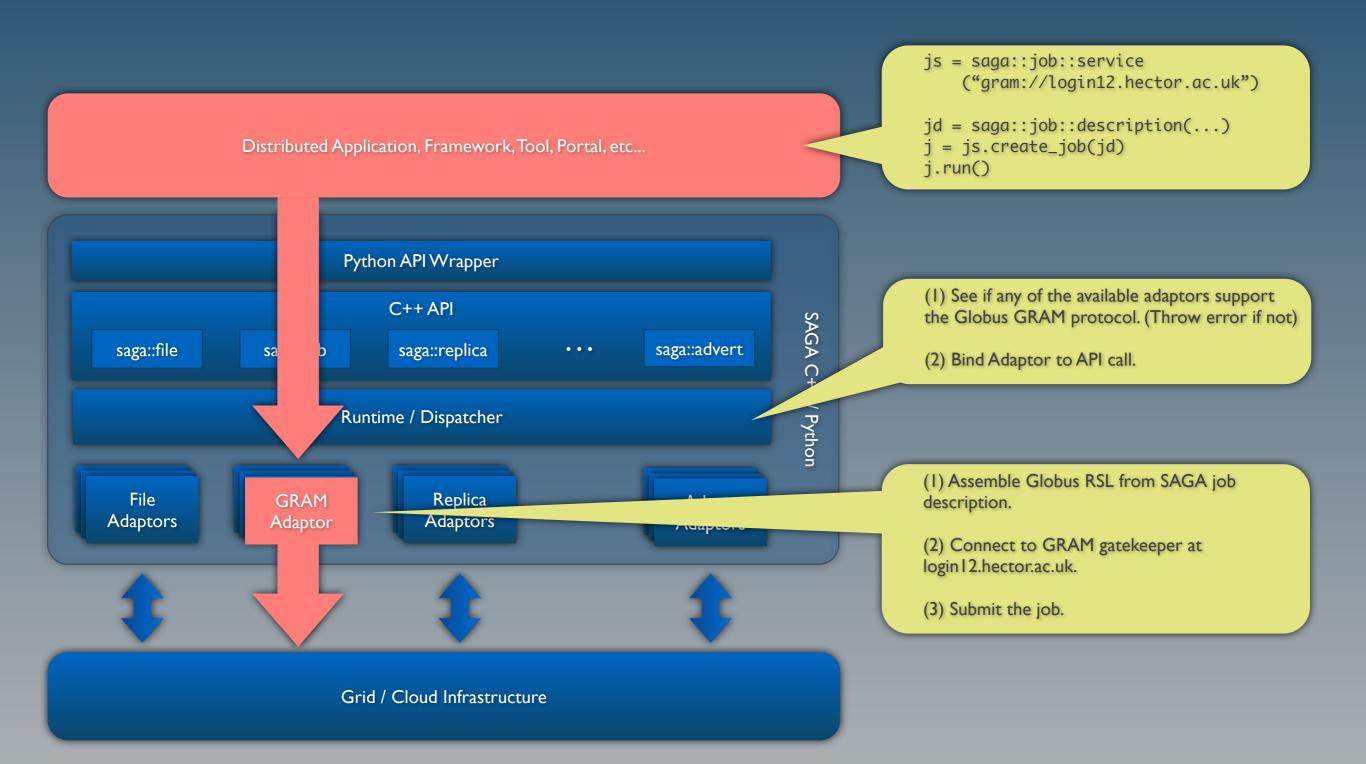




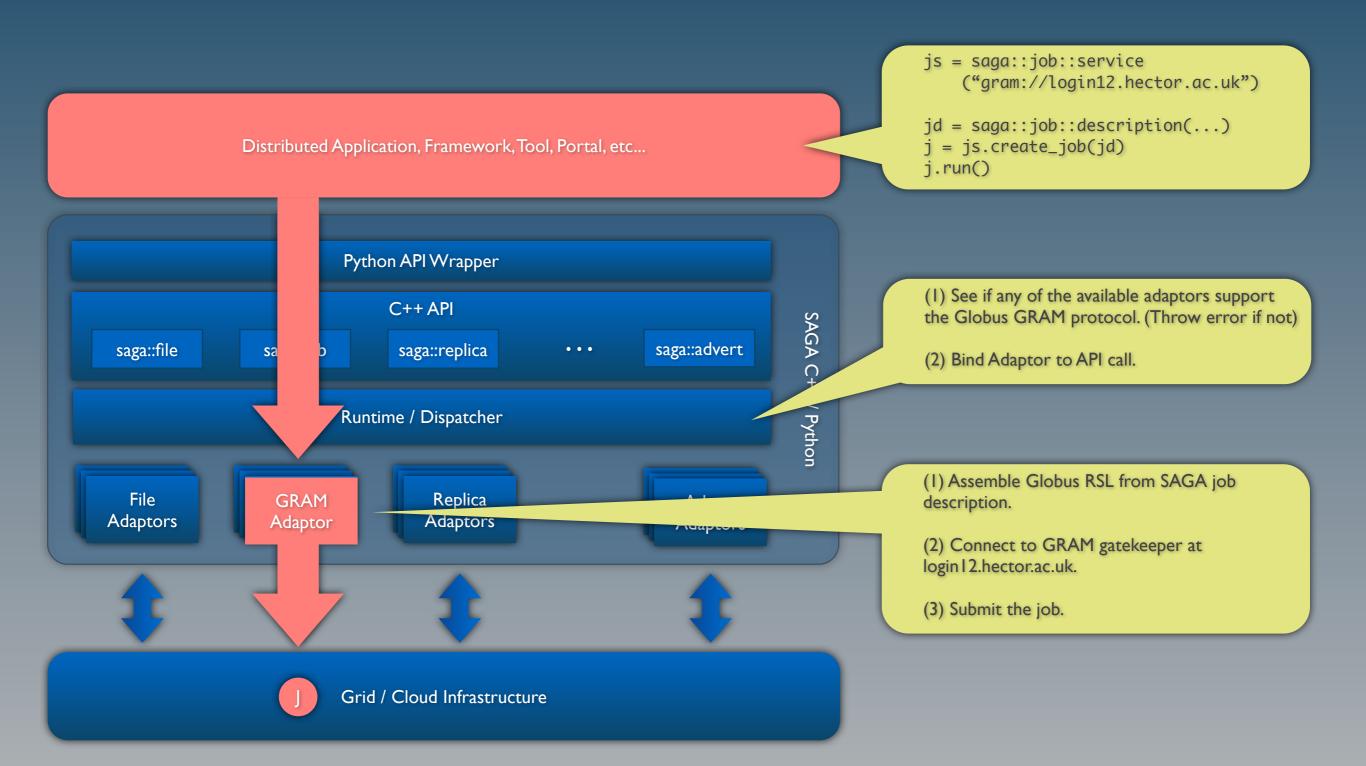












API Packages



- saga::advert Advert Service Access API
- saga::filesystem Filesystem Access API
- saga::job Job Submission & Management API
- saga::replica Replica Catalog Management API
- saga::rpc Remote Procedure Call API
- saga::sd Service Discovery API
- saga::stream Data Stream Client & Server
- API Extensions under development: saga::messaging



- saga::job Arc, Amazon EC2, Condor, Eucalyptus, Globus GRAM (2&5), Fork, gLite, SSH, Nimubs, OGSA BES, PBS (Pro), Platform LSF, SSH, TORQUE, ...
- saga::filesystem Globus GridFTP, Hadoop HDFS, Local Filesystem, SSHFS, ...
- saga::replica Globus RLS, SQL Replica Service
- saga::advert SQL Advert Service
- saga::stream TCP-based
- saga::sd gLite SD



- After several years of prototyping, testing and hardening, we presented SAGA as the holy grail of distributed computing to the user communities
- Lots of advertising, demos, workshops, tutorials
- But: Uptake very slow and not as expected
- Almost by accident, the problem got solved with the "SAGA Big-Job" framework

A Small Dilemma ?



# gridftpClientToServer.C #		Class : GridFTP
# # Description : Transfer the data file from Client site to each remote site. #		Function : GridFTP (Constructor of class GridFTP)
# and return the status of the file transfer. # # #		Description : Used to a) Activate the GridFTP client module
# Input : Source URL string of the source server , and the Destination URL # # string of the destination server . #		b) Initialize the mutex lock and condition variables c) Initialize the GridFTP client handle
# The format for URL string is # # Source URL : full path of the file to transferred #		
<pre># Destination URL : gsiftp://server-hostname/fullpathname/ofthefile/tobetransfered #</pre>		GridFTP::GridFTP() {
# 0utput : Displays whether the transfer is SUCCESSFUL/FAILED. # # # #	o h	<pre>buffer_length = MAX_BUFFER_SIZE; status = (globus_result_)globus_module_activate(GLOBUS_FTP_CLIENT_MODULE);</pre>
ff -/		if (status != GLOBUS_SUCCESS) (
#include <fcntl.h></fcntl.h>		<pre>tmpstr = globus_object_printable_to_string(globus_error_get(status));</pre>
#include*iomanip"		cout<<'\n\E Error: Failed to load GLOBUS_FTP_CLIENT_MODULE.\n\t Error Code "< <status<<"\n\t"<ttmpstr<<endl; exit(1);</status<<"\n\t"<ttmpstr<<endl;
<pre>#include"globus_ftp_client.h" #include"globus_common.h"</pre>	mm	3
#include <iostream></iostream>); /
#includestring> #includestrem>		Class : GridTTP
#include*common.C"		Function : -GridFTP (Destructor of class GridFTP)
#include"globusCallback.C" using namespace std;		Description : Used to a) Destroy the GridFTP client handle
//FILE *fd; #define Max SuppEr SIZE 2048		b) Deactivate the Grid#TP client module
#define ERROR =1		GridFTP::-GridFTP() {
		<pre>globus_module_deactivate_all();</pre>
class GridFTP:public InputConfig,public GlobusCallback {		}; /*
globus_ftp_client_handleattr_t hattr; globus_ftp_client_operationattr_t oattr;		CLass : GridFTP Function : gridftpClientToServer()
globus_ftp_client_handle_t handle; globus_byte_t buffer[MAX_BUFFER_SIZE];		Description : To transfer the file from client site to remote site Input : a) Source file path which is to transferred
globus_size_t buffer_length; globus_result_t status;		b) Destination url to store the file c) LogFile
char • tmpstr;		
<pre>public : GridFTP();</pre>	no	int GridFTP::gridftpClientToServer(string src,string dst) {
-GridFTP(); int gridftpClientToServer(string ,string);		int rc; FILE *fd;
};		/* Initialize the handle attribute */
//		<pre>if (globus ftp_client_handleattr_init(shattr) != GLOBUS_SUCCESS) { cout<<"\n\t\t ERROR : Failed to activate the ftp client handleattr\n";</pre>
<pre> Function : done cb()</pre>		return 1;
Used : gridftpClientToServer(),gridftpThirdPartyTransfer(),gridftpServerToClient()		/* Initialize the operation attribute */
Description : It is a callback function ,it is called when the transfer is	i ohl	<pre>if (globus_ftp_client_operationattr_int(soattr) != GLOBUS_SUCCESS) { cout<<*\n\t\t ERROR : Failed to initialize operationattr\n";</pre>
completely finished, i.e. both the data channel and control channel exchange.		return 1;
Here it simply sets a global variable (done) to true so the main		
program will exit the while loop. */		<pre>/* Initalize the handle */ if (globus_ftp_client_handle_init(&handle,&hattr) != GLOBUS_SUCCESS) { cont<<`\n\t\t ERROR : Failed to initialize the handle\n';</pre>
namespace gridftp_cb {		return 1;
<pre>static void dome_cb (void *user_arg , globus_ftp_client_handle_t *handle , globus_object_t *err) {</pre>		3
GridFTP *monitor=(GridFTP*)user_arg;		continueOnCond();
char * tmpstr;		<pre>fd = fopen(src.c_str(), *r"); if (fd == NULL) {</pre>
<pre>if (err){ cout<<"\t\t Status : File Transferred Failed "<<endl;< pre=""></endl;<></pre>		<pre>cout<"Error in opening local file"<<src; return 1;</src; </pre>
<pre>cout<<*\t\t ERROR :*<<globus_object_printable_to_string(err)<<endl; pre="" }<=""></globus_object_printable_to_string(err)<<endl;></pre>		3
else cout<<"\t\t Status : File Transferred Successfully*< <endl;< td=""><td></td><td><pre>/* Gridftp API call to start the put operation */ status = globus_ftp_client_put(ikandle,dst.c_str(),GLOBUS_NULL,GLOBUS_NULL,gridftp_cb::done_cb,this);</pre></td></endl;<>		<pre>/* Gridftp API call to start the put operation */ status = globus_ftp_client_put(ikandle,dst.c_str(),GLOBUS_NULL,GLOBUS_NULL,gridftp_cb::done_cb,this);</pre>
<pre>monitor->setDoneValue();</pre>		if (status != GLOBUS_SUCCESS) {
return; };		<pre>globus_object_t * err; err = globus_error_get(status);</pre>
/*		cout< <endl; fprintf(stderr, "\tError : %s", globus_object_printable_to_string(err));</endl;
Function : data_cb()		done = GLOBUS_TRUE; }
Used : gridftpClientToServer()		else { rc = fread(buffer,1,MAX_BUFFER_SIZE,fd);
Description : read or write operation in the FTP Client library is asynchronous.A		globus_ftp_client_register_write(
callback of this type is passed to such data operation function calls. It is called when the user supplied buffer has been successfully		buffer, rc,
transferred to the kernel. Note: That does not mean it has been successfully transmitted,instead it		0, feof(fd) i= success,
just reads the next block of data and calls register_write/register_read again. */		gridftp_cb::data_cb, (void *) fd);
static void data_cb (void *user_arg , globus_ftp_client_handle_t *handle,globus_object_t * err, globus_byte_t *buffer,		}
<pre>globus_size_t length , globus_off_t offset,globus_bool_t eof) {</pre>		/* lock on condition */ waitOnCond();
<pre>if (err) { cout<<"\t\t ERROR :"<<globus_object_printable_to_string(err)<<endl;< pre=""></globus_object_printable_to_string(err)<<endl;<></pre>		fclose(fd);
) else {		<pre>globus_ftp_client_handle_destroy(&handle); return 0;</pre>
<pre>if (leof) { FILE *fd = (FILE *) user_arg;</pre>); ;
int rc; rc = fread(buffer, 1, MAX_BUFFER_SIZE, fd);		/* start of main */
<pre>if (ferror(fd) != \$UCCESS) { cout<<*\n\t\t Error : Function data_cb\n*<<*\t\t Error code :*<< errno<<endl;< pre=""></endl;<></pre>		int main(int argc , char **argv) {
return; }		int hostCount; string destinationUrl,sourceUrl,tempUrl;
<pre>globus_ftp_client_register_write(handle, buffer,rc,offset + length,feof(fd) != SUCCESS, } data_cb, (void *) fd); }</pre>		string sourceFile, destinationFile, outputFile;
) return;		/* Creating the object */ GridFTP gridftp;
);		
e.weigher(weg.ac.uk		

A Small Dilemma ?



<pre></pre>		
<pre>indications is in a constraint of the definition of the defin</pre>	1*	/+
<pre> i</pre>	# # gridftpClientToServer.C	#
<pre>i arrow i arrow i</pre>		ŧ.
<pre> def de de</pre>	#	# a) Activate the GridFTP client module
<pre> i but due is in the state and in the state and is in the sta</pre>	# string of the destination server .	
<pre> index device to remain the remain the</pre>	# Source URL : full path of the file to transferred	
<pre> iiiiii</pre>	#	
Rink relations Image: Second relation relation relations Image: Second rel	<pre># Output : Displays whether the transfer is SUCCESSFUL/FAILED. # #</pre>	
An under sufficiency	# */	if (status 1= GLOBUS_SUCCESS) {
Image: Second		<pre>tmpstr = globus_object_printable_to_string(globus_error_get(status));</pre>
Transform		
All of the second o		
Calaboration Calaboration <td></td> <td></td>		
<pre> in the second sec</pre>		
In the second secon	#include"common.C"	
All provide a constraints <td>using namespace std;</td> <td>a) Destroy the GridFTP Client handle</td>	using namespace std;	a) Destroy the GridFTP Client handle
And ENDINE And Endines And En	#define MAX_BUFFER_SIZE 2048	b) Deactivate the GridFTP client module
<pre>table definition function function</pre>		
with the state stat		/*
in the first status status in the first status of the file transfer. # in the first status of the file transfer. # in the first status of the file transfer. # in the first status of the data file from client site to each remote site # in the first status of the data file from client site to each remote site # in the first status of the data file from client site to each remote site # in the first status of the destination server . # in the first status of the destination server . # in the first status of the destination server . # in the first status of the destination server . # in the first status of the destination server . # in the first status of the destination server . # in the first status of the destination server . # in the first status of the destination server . # in the first status of the destination server . # in the first status of the destination server . # in the first status of the destination server . # in the first status of the destination server . # in the first status of the destination server . # in the first status of the destination server . # in the first status of the destination server . # in the first status of the destination server . # in the first status of the destination server . # in the first status of the destination server . # in the first status of the first status of the first	#	
<pre> the start is the start is the start is the start is of the file transfer.</pre>	globus_ftp_client_operationattr_t oattr;	
<pre>def</pre>	globus_byte_t buffer[//	griditpClientToServer.C #
<pre>puis to dear function to the data file from client site to each remote site // * Description : Transfer the data file from client site to each remote site // * Description : Transfer the data file from client site to each remote site // * // * // * // * // * // * // * //</pre>	globus_result_t status; #	
<pre>minume manual manu</pre>	# Descript	ion: Transfer the data file from client site to each remote site #
Image: State	a filmen ()	
<pre>// ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ ** */ */</pre>	-GridFTP(); ##	and return the status of the file transfer. #
<pre>Runt : :m</pre>	»,	+
<pre>Runt : :m</pre>	/* # Tnput	: Source URL string of the source server, and the Destination URL #
<pre>und i printpilementer(, printpilementer(</pre>		
<pre>source uses is the set of th</pre>	Used : gridftpClientToServer(),gridftpThirdPar	
<pre>setting:::::::::::::::::::::::::::::::::::</pre>		The format for URL string is #
<pre>more will and we like large</pre>	exchange. #	Source URL : full path of the file to transferred #
<pre>amages up up</pre>	Here it simply sets a global variable (dor program will exit the while loop. #	
<pre>state valdes_db (vald **ser_ser_ser_ser_ser_ser_ser_ser_ser_ser_</pre>	<i>π</i>	$= \frac{1}{2} = $
<pre># Output : Displays whether the transfer is SUCCESSFUL/FAILED. # if output : Displays whether the transfer is SUCCESSFUL/FAILED. # if output : for if o</pre>	"	#
<pre>uniter "Books (uniter") set upsy:</pre>		: Displays whether the transfer is SUCCESSFUL/FAILED. #
<pre>if (err) contex'hit status : plin contex contex contex contex contex contex contex contex contex co</pre>	- <u>#</u>	#
<pre>contex='tit's EEOR i "cedind "</pre>	if (err){	
<pre>contex*'tht Status : File1 contex*'tht Stat</pre>		#
<pre>suitor->setDoneValue(); return; ;; return; ;; /*</pre>		*/
<pre>return; }; return; ;; /*</pre>		
/*	return;	
Punction : data_cb() dome = GLOBUS_TRUE; Used : gridtpClientToServer() else {	};	cout< <endl;< td=""></endl;<>
Function : dat_cb() } Used : gridftpClientOserver() else { c : gridftpClientToserver() c : freed(buffer,1,MAX_BUFFER_SIZE,fd); globus_ftp_client_register_vrite(/	
<pre>gescription : read or write operation in the PTP Client library is asynchronous.A callback of this type is passed to such data operation function calls. callback of this type is passed to such data operation function calls. Li is called when the user supplied buffer has been successfully transmitted, instead it just reads the next block of data and calls register_write/register_read again</pre>) else (
bescription : read or write operation in the PPP Client library is asynchronous.A ishandle, callback of this type in passed to such data operation function calls. buffer, It is called when the user supplied buffer has been successfully rc, Transferred to the kernel. 0, Note: That does not mean it has been successfully transmitted, instead it feof[d] != SUCCESS,		globus_ftp_client_register_write(
transferred to the kernel. Note: That does not mean it has been successfully transmitted, instead it just reads the next block of data and calls register_write/register_read again. 	callback of this type is passed to such data operation function calls.	ahandle, buffer,
just reads the next block of data and calls register_write/register_read again. 	It is called when the user supplied buffer has been successfully transferred to the kernel.	0,
static void data_cb (void *user_arg , globus_ftp_client_handle_t *handle,globus_object_t * err, globus_byte_t *buffer, globus_size_t length , globus_off_t offset,globus_bool_t eof) /* lock on condition */	just reads the next block of data and calls register_write/register_read again.	gridftp_cb::data_cb,
globus_size_t length , globus_off_t offset,globus_bool_t eof) /* lock on condition */		
<pre>waitOnCond();</pre>	<pre>static void data_cb (void *user_arg , globus_ftp_client_handle_t *handle,globus_object_t * er globus_size_t length , globus_off_t offset,globus_bool_t eof)</pre>	/* lock on condition */
if (err) { fclose(fd);	{ if (err) {	
cout<"\t\t ERROR :"< <globus_object_printable_to_string(err)<<endl; globus_ftp_client_handle_destroy(thandle);<="" j="" td=""><td><pre>cout<<"\t\t ERROR :"<<globus_object_printable_to_string(err)<<endl; pre="" }<=""></globus_object_printable_to_string(err)<<endl;></pre></td><td><pre>globus_ftp_client_handle_destroy(6handle);</pre></td></globus_object_printable_to_string(err)<<endl;>	<pre>cout<<"\t\t ERROR :"<<globus_object_printable_to_string(err)<<endl; pre="" }<=""></globus_object_printable_to_string(err)<<endl;></pre>	<pre>globus_ftp_client_handle_destroy(6handle);</pre>
else {	if (leof) {	return 0;
<pre>FILE #fd = (FILE *) user_arg; int rc;</pre>	<pre>FILE *fd = (FILE *) user_arg; int rc;</pre>	
rc = fread(buffer, 1, MAX_BUFFER_SIZE, fd); if (ferror(fd) 1 = SUCCES) {	<pre>rc = fread(buffer, 1, MAX_BUFFER_SIZE, fd); if (ferror(fd) != SUCCESS) {</pre>	<pre>int main(int argc , char **argv) {</pre>
cout<<'\n\t\t Error : Function data_cb\n*<<'\t\t Error code :*<< errno< <endl; return; int hostCount;</endl; 	cout<"\n\t\t Error : Function data_cb\n"<<"\t\t Error code	:"< <pre>rmo<endl;< pre=""></endl;<></pre>
<pre>} string destinationUrl,sourceUrl,tempUrl; globus_ftp_client_register_write(handle, buffer,rc,offset + length,feof(fd) != SUCCESS, data_cb, (void *) fd); string sourceFile,destinationFile,outputFile;</pre>	}	string destinationUrl, sourceUrl, tempUrl;
} /* Creating the object */	}	
return; GridFTP gridftp;		
	return;);	

A Small Dilemma ?



1*	/*
# gridftpClientToServer.C #	Class : GridFTP
## # Description : Transfer the data file from client site to each remote site. #	Function : GridFTP (Constructor of class GridFTP)
# and return the status of the file transfer. # # #	Postription : Used to a) Activate the GridFTP client module
# Input : Source URL string of the source server , and the Destination URL # # string of the destination server . #	b) Initialize the mutex lock and condition variables c) Initialize the GridFTP client handle
# The format for URL string is # # Source URL : rull path of the file to transferred #	•/
# Destination URL : gsiftp://server-hostname/fullpathname/ofthefile/tobetransfered #	GridPTP::GridPTP() {
# Output : Displays whether the transfer is SUCCESSFUL/FAILED. #	buffer_length = MAX_BUFFER_SIZE; status = (globus_result_t)globus_module_activate(GLOBUS_FTP_CLIENT_MODULE);
¢	<pre>status = (globus_result_t)globus_module_activate(GLOBUS_FTP_CLIENT_MODULE); if (status 1= GLOBUS_SUCCESS) {</pre>
#include <fcntl.h></fcntl.h>	<pre>tmpstr = globus_object_printable_to_string(globus_error_get(status));</pre>
#include'iomanip"	<pre>cout<<^\ht Error: Failed to load GLOBUS_FTP_CLIENT_MODULE.\h\t Error Code *<<status<<*\h\t+*<<tmpstr<<endl; exit(1);</status<<*\h\t+*<<tmpstr<<endl; </pre>
<pre>#include"globus_ftp_client.h" #include"globus_common.h"</pre>	
finclude <iostream></iostream>	2
<pre>#include≤trinp> #include≤trinp></pre>	Class : GridFTP
≠include*common.C*	Function : -GridFTP (Destructor of class GridFTP)
#include "globusCallback.C" using namespace std;	Description : Used to a) Destroy the GridFTP client handle
//FILE *fd; #define MAX BUFFER SIZE 2048	b) Deactivate the GridFTP client module
#define ERROR -1 #define SUCCESS 0	
class GridFTP:public InputConfig,public GlobusCallback {	
globus_ftp_client_handleattr_t hattr; # #include <saga saga.hpp=""></saga>	#
globus ftp client handle_t handle; # # #in clude <i cm<="" gt="" no="" o="" td=""><td></td></i>	
globus size t buffer "	
globus_result_t status; # char * tmpstr; //	
<pre># Des(int main (int argc, char** argv)</pre>	
-GridFTP(); #	#
int gridrtpcilentroserver(string, string);	
/* # Inpl saga::filesystem::file f(std::s	string(argv[1])); #
$# \qquad f copy(std:string(argy[2]))$	
#	
Description : It is a callback function , it is called wh # } }	
completely finished, i.e. both the data ct exchange. Here it simply sets a global variable (dor # catch(saga::exception const & e)	
completely finished, i.e. both the data ct //	# # ad #
<pre>completely finished, i.e. both the data ct exchange. Here it simply sets a global variable (dor program will exit the while loop. # " " " " " " " " " " " " " " " " " "</pre>	
<pre>completely finished, i.e. both the data ct exchange. Here it simply sets a global variable (dor program will exit the while loop. namespace gridftp_cb { # std::cerr << "Error: " << e.what } </pre>	
<pre>completely finished, i.e. both the data ct exchange. Bere it simply sets a global variable (dor program will exit the while loop. # namespace gridftp_cb (# std::cerr << "Error: " << e.what</pre>	
<pre>completely finished, i.e. both the data ct exchange. Here it simply sets a global variable (dor program vill exit the while loop. namespace gridftp_cb { static void done_cb (void *user_arg, glc gridftp *monitor=(GridfTP*)user_arg; char * tmpstr; # } Catch(saga::exception const & e) { std::cerr << "Error: " << e.what } } </pre>	
<pre>completely finished, i.e. both the data ct exchange. Bere it simply sets a global variable (dor program will exit the while loop. namespace gridftp_cb { static void done_cb (void *user_arg , glc fi (err) { cout<<*\th Status : File 1 # } } cut<<*\th Status : File 1 # } // cut<<*/> // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // //</pre>	
<pre>completely finished, i.e. both the data ct exchange. Here it simply sets a global variable (dor program vill exit the while loop. namespace gridftp_cb { static void dome_cb (void *user_arg , glc fidfTP *monitor=(GridFTP*)user_arg; char * tmpstr; if (err }{ cout<<*\t\t Status : File 7 cout<<*\t\t BEROR :*<<qlabel{eq:global td="" variable}<=""><td></td></qlabel{eq:global></pre>	
<pre>completely finished, i.e. both the data ct exchange. Bere it simply sets a global variable (dor program will exit the while loop. namespace gridftp_cb { static void done_cb (void *user_arg , glc fi (err) { cout<<*\th Status : File 1 # } } cut<<*\th Status : File 1 # } // cut<<*/> // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // // //</pre>	
<pre>completely finished, i.e. both the data ct exhamps mere it simply sets a global variable (dor program will exit the while loop. namespace gridftp_cb { static void done_cb (void *user_arg , glc fi (err) { cout<<*\t\t Status : File 1 cout<<*\t\t Status : File 1 monitor->setDoneValue(); } } char * tmpstr; if (err) { cout<<*\t\t Status : File 1 monitor->setDoneValue(); } char * cout<<*\t\t Status : File 1 monitor->setDoneValue(); } char * cout<<*\t\t Status : File 1 monitor - SetDoneValue(); } char * cout<<*\t\t Status : File 1 monitor - SetDoneValue(); } char * cout<<*\t\t Status : File 1 monitor - SetDoneValue(); } char * cout<<*\t\t Status : File 1 monitor - SetDoneValue(); } char * cout<<*\there is a global variable of the data ct</pre>	t() << std::endl;
<pre>schappen completely finished, i.e. both the data ct schampe. Bere it simply sets a global variable (dor program vill exit the while loop. namespace gridftp_cb { static void dome_cb { void *user_arg , glc fridFTP *monitor=(GridFTP*)user_arg; char * tmgstr; if { err }{ cout<<<`\t\t Status : File * cout<<<`\</pre>	<pre>t() << std::endl;</pre>
<pre>scharge. mamespace gridftp.cb { static void dome_cb (void *user_arg , glc dridfT* *monitor=(GridfT*)user_arg; clar * tmpstr; if (err) { cout<<^\tt Status : File 1 else cout<<^\tt Status : File 1 monitor-setDoneValue(); return; return; } } context in the data ct # context the status : File 1 monitor-setDoneValue(); return; context the data ct # context the data ct for the data ct for the data ct context the data ct for the data ct for the data ct context the data ct for the data ct for</pre>	<pre>t() << std::endl; # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #</pre>
<pre>scharge. mamespace gridftp.cb { static void dome_cb (void *user_arg , glc dridfT* *monitor=(GridfT*)user_arg; clar * tmpstr; if (err) { cout<<^\tt Status : File 1 else cout<<^\tt Status : File 1 monitor-setDoneValue(); return; return; } } context in the data ct # context the status : File 1 monitor-setDoneValue(); return; context the data ct # context the data ct for the data ct for the data ct context the data ct for the data ct for the data ct context the data ct for the data ct for</pre>	<pre>t() << std::endl; # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #</pre>
<pre>completely finished, i.e. both the data ct schampe. manage. m</pre>	<pre>t() << std::endl; # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #</pre>
<pre>completely finished, i.e. both the data of sexhange. Bere it simply sets a global variable (dor program will exit the while loop. namespace gridtp_cb { static void dome_cb (void *user_arg, glc GridTP *monitor=(GridTP*)user_arg; char * tmpstr; if (err } { cout<<'\th Status : File 1 monitor->setDoneValue(); return; }; /*</pre>	<pre>t() << std::endl; # # # # # # # # # # # # # # # # # # #</pre>
<pre>completely finished, i.e. both the data of sexhampy. Bere it simply sets a global variable (dor program will exit the while loop. namespace gridtp_cb { static void dome_cb { void *user_arg , glc driftTP *monitor=(GridTTP*)user_arg; char * tmpstr; if { err } { cout<<`\tt Status : File 1 monitor->setDoneValue(); return; }; /* Function : data_cb() Used : gridtpClientToServer() Description : read or write operation in the FTP Client library is asynchronous.A callback of this type is passed to such data operation function calls. t is called when the user supplied buffer has been successfully the super course of the super course.a callback of this type is passed to such data operation function calls. t is called when the user supplied buffer has been successfully the super course operation in the super course.A callback of this type is passed to such data operation function calls. t is called when the user supplied buffer has been successfully the supplied buffer has been successfully the super course operation in the super course.a callback of this type is passed to such data operation function calls. t is called when the user supplied buffer has been successfully the supplied buffer has been successfully the super course operation is the super course operation function calls. t is called when the supplied buffer has been successfully the supplied buffer has been successfully</pre>	<pre>t() << std::endl; # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #</pre>
<pre>completely finished, i.e. both the data of secange. Here it simply sets a global variable (do program will exit the while loop. namespace gridftp_cb { static void dome_cb (void *user_arg , gle dridfTP *monitore(GridfTP*)user_arg; char * tmpstr; if (err) { cont<<*/tyle HERM :*</pre>	<pre>t() << std::endl; # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #</pre>
<pre>/*</pre>	<pre>t() << std::endl; # # # # # # # # # # # # # # # # # # #</pre>
<pre>completely finished, i.e. both the data of schampe. Bere it simply sets a global variable (dor program vill exit the while loop. namespace gridftp_cb { static void done_cb (void *user_arg, glc dridFTP *monitor-(GridFTP*)user_arg; char * tangatr; if (err){ cout< #</pre>	<pre>t() << std::endl;</pre>
<pre>completely finished, i.e. both the data of exchange. Here it singly sets a global variable (do program vill exit the while loop. namespace gridftp_cb { static rooid done_cb (void *user_arg , glo globa_size_t length , globa_stp_client_handle_t *handle.globa_sblect_t * err, globa_byte_t *buffer, globa_size_t length , globa_stp_client_handle_t *handle.globa_sblect_t * err, globa_byte_t *buffer, globa_size_t length , globa_solt_t offset, globa_bool_t eof) for the state of the state o</pre>	<pre>t() << std::endl; # # # # # # # # # # # # # # # # # #</pre>
<pre>completely finished, i.e. both the data day compared with each one of the system of the system</pre>	<pre>t() << std::endl;</pre>
<pre>complexity finished, i.e. both the data d</pre>	<pre>t() << std::endl; # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #</pre>
<pre>completely finished, i.e. both the data of</pre>	<pre>t() << std::endl; # # # # # # # # # # # # # # # # #</pre>
<pre>complety finished is. both the data of sechage. Here is imply sets a global variable (dec program ville stat the while loop. # for white value is the while loop. # for der value value of (didTP*)user_arg; char + Uspur; if (err) { contex(*tht Balaus : File i</pre>	<pre>t() << std::endl;</pre>
<pre>compared is included in the base is a gradient of the set of</pre>	<pre>t() << std::endl; # # # # # # # # # # # # # # # # # # #</pre>
<pre>compared in initial is is both the data at sechange Bere it simply sets a global variable (der groups will and it the while hop.</pre>	<pre>t() << std::endl; # # # # # # # # # # # # # # # # # # #</pre>
<pre>compared is included in the base is a grant of the data of a schemps. Base it simply set a global variable (det mamspace gridfug,cb { static wold dome,cb { wold "user_arg ; global fif { erg } { fif { erg } {</pre>	<pre>t() << std::endl; # # # # # # # # # # # # # # # # # # #</pre>
<pre>completely finished, i.e. both the date of</pre>	<pre>t() << std::endl; # # # # # # # # # # # # # # # # # # #</pre>
<pre>completity filled, i.e. both the date d</pre>	<pre>t() << std::endl; # # # # # # # # # # # # # # # # # # #</pre>
<pre>complexity finished, i.e. both the data of</pre>	<pre>t() << std::endl; # # # # # # # # # # # # # # # # # # #</pre>



- After several years of prototyping, testing and hardening, we presented SAGA as the holy grail of distributed computing to the user communities
- Lots of advertising, demos, workshops, tutorials
- But: Uptake very slow and not as expected
- Almost by accident, the problem got solved with the "SAGA Big-Job" framework



- Big-Job (a.k.a. glide-in, or pilot-job) is a simple distributed programming abstraction written in SAGA (Python)
- It allows to run lots of HTC jobs (often single-core) transparently on HPC machines using overlay scheduling



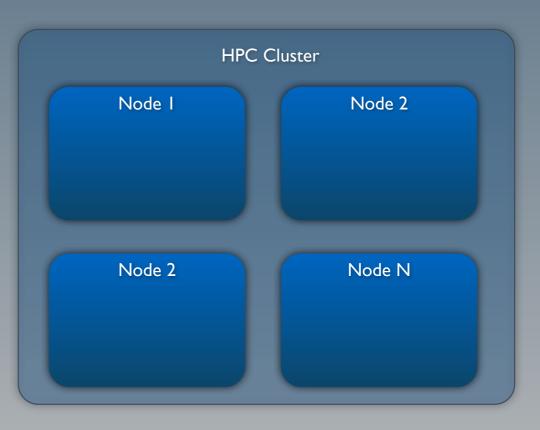
- Big-Job (a.k.a. glide-in, or pilot-job) is a simple distributed programming abstraction written in SAGA (Python)
- It allows to run lots of HTC jobs (often single-core) transparently on HPC machines using overlay scheduling





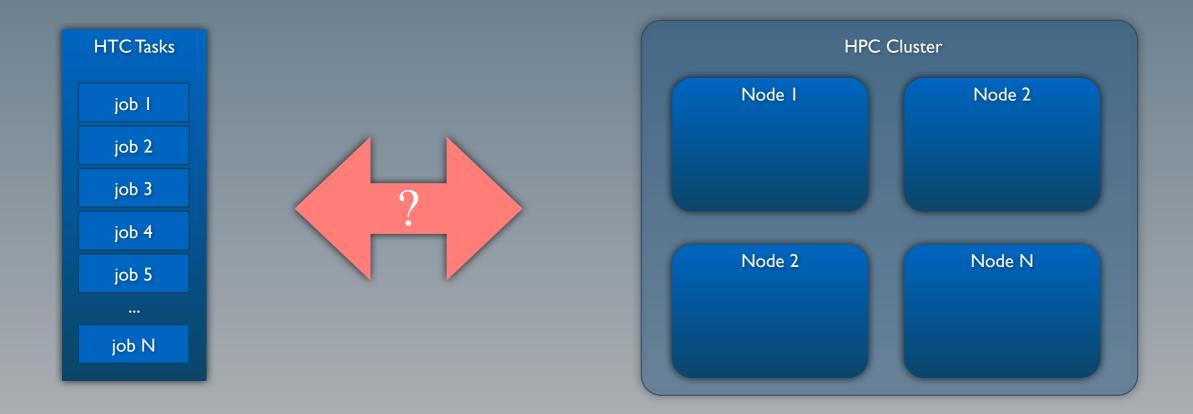
- Big-Job (a.k.a. glide-in, or pilot-job) is a simple distributed programming abstraction written in SAGA (Python)
- It allows to run lots of HTC jobs (often single-core) transparently on HPC machines using overlay scheduling





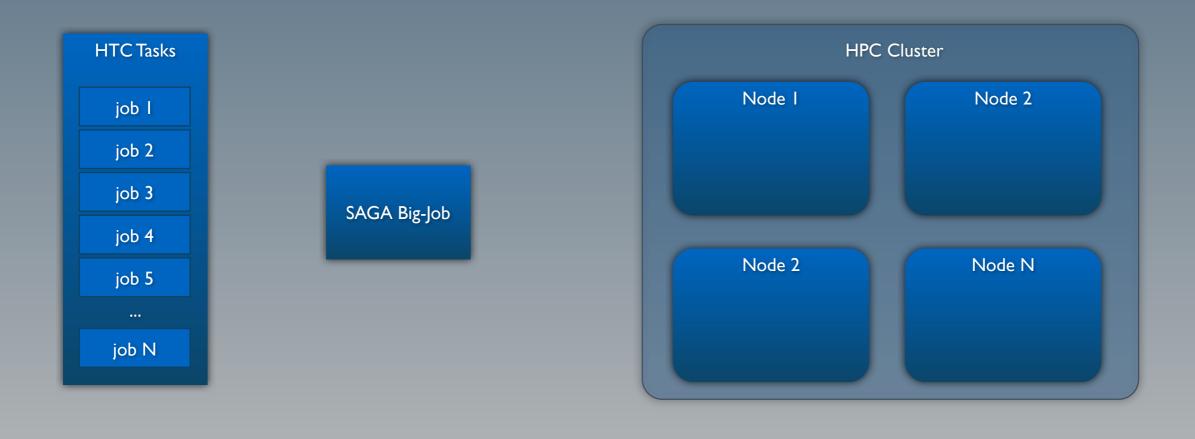


- Big-Job (a.k.a. glide-in, or pilot-job) is a simple distributed programming abstraction written in SAGA (Python)
- It allows to run lots of HTC jobs (often single-core) transparently on HPC machines using overlay scheduling



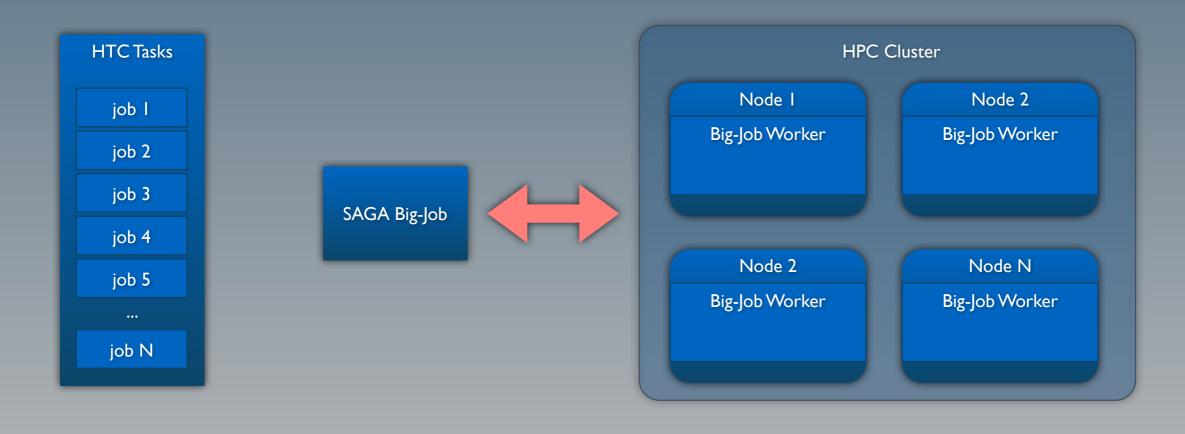


- Big-Job (a.k.a. glide-in, or pilot-job) is a simple distributed programming abstraction written in SAGA (Python)
- It allows to run lots of HTC jobs (often single-core) transparently on HPC machines using overlay scheduling



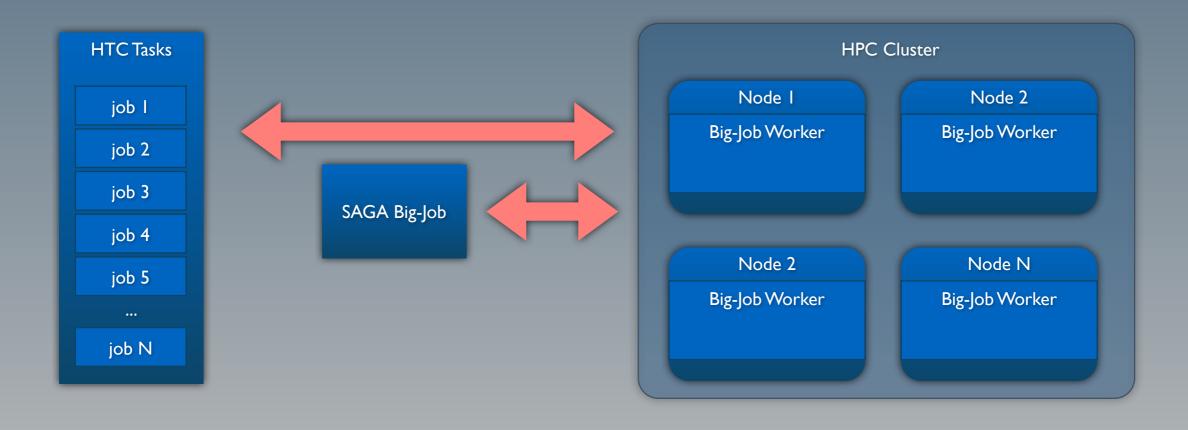


- Big-Job (a.k.a. glide-in, or pilot-job) is a simple distributed programming abstraction written in SAGA (Python)
- It allows to run lots of HTC jobs (often single-core) transparently on HPC machines using overlay scheduling



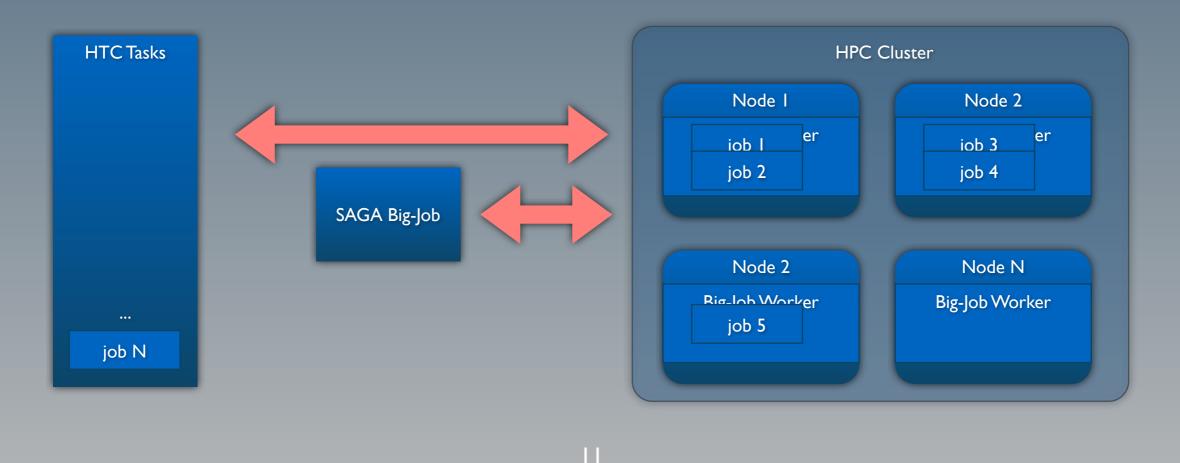


- Big-Job (a.k.a. glide-in, or pilot-job) is a simple distributed programming abstraction written in SAGA (Python)
- It allows to run lots of HTC jobs (often single-core) transparently on HPC machines using overlay scheduling





- Big-Job (a.k.a. glide-in, or pilot-job) is a simple distributed programming abstraction written in SAGA (Python)
- It allows to run lots of HTC jobs (often single-core) transparently on HPC machines using overlay scheduling





- Big-Job (a.k.a. glide-in, or pilot-job) is a simple distributed programming abstraction written in SAGA (Python)
- It allows to run lots of HTC jobs (often single-core) transparently on HPC machines using overlay scheduling
- Big-Job suddenly sparked a lot of interest in SAGA
 - Makes a huge problem simply disappear
 - Allows to use legacy code (non-intrusive)
 - Runs everywhere even on Hector (CRAY) and EC2

Abstractions, Abstractions, Abstractions !



- Apparently, we didn't understand the user communities and their requirements properly
- Users (especially the non-technical users) don't want another API. They want simple solutions for their every day problems. Abstractions can help !
- It turns out that SAGA is perfect to develop distributed programming abstractions:
 - Hides specific middleware implementation details
 - Allows concurrent cross-infrastructure resource usage
 - Optimisation and "adaptation" can happen "behind the scenes"

Abstractions, Abstractions, Abstractions !



• Abstractions: from generic to specialised:

- Pilot-Job (a.k.a. Big-job, a.k.a. Glide-in)
- Pilot-Data (data affinity)
- Master-Worker
- Workflow (a.k.a. DAG)
- Peer-to-Peer
- Replica exchange
- Map-Reduce
- All-Pairs
- See eSI 3DPAS theme: <u>http://www.esi.ac.uk/research-themes/5</u>

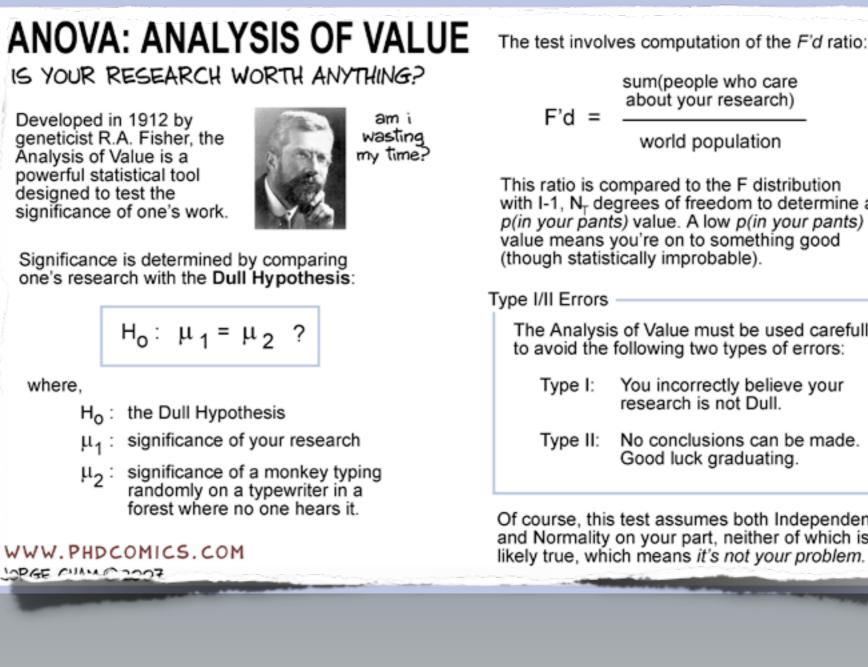
Abstractions, Abstractions, Abstractions !



- Abstractions: from generic to specialised: Pilot-Job (a.k.a. Big-job, a.k.a. Glide-in) Pilot-Data (data affinity) Master-Worker •) Workflow (a.k.a. DAG) Peer-to-Peer) Replica exchange Map-Reduce All-Pairs
 - See eSI 3DPAS theme: <u>http://www.esi.ac.uk/research-themes/5</u>

Is There Potential for (PhD) Research ?





The test involves computation of the F'd ratio: sum(people who care about your research) F'd =world population This ratio is compared to the F distribution with I-1, N_r degrees of freedom to determine a p(in your pants) value. A low p(in your pants) value means you're on to something good (though statistically improbable). Type I/II Errors The Analysis of Value must be used carefully to avoid the following two types of errors: You incorrectly believe your Type I: research is not Dull. Type II: No conclusions can be made. Good luck graduating. Of course, this test assumes both Independence and Normality on your part, neither of which is

Is There Potential for (PhD) Research ?



- Previous work and experience has shown the relevance and importance of abstractions in distributed computing
 - Especially interesting: with abstractions, optimisation can be done "behind the scenes" (hidden from the user)
- Some fundamental questions:
 - What are interesting upcoming challenges in future distributed applications and systems ?
 - Can some of the challenges possibly be address by using distributed programming abstractions ?
 - What is the current state of research in distributed programming abstractions ?

What are the Upcoming Challenges ?



• The Data Deluge

- "The amount of genomic data available for study is increasing at a rate similar to that of Moore's Law" http://www.mcs.anl.gov/uploads/cels/papers/P1238.pdf
- "Store now process later" might not always be an option
- Adaptive algorithms will be required to handle vast amounts of dynamic (streaming) data
- Methods to characterise/predict data and especially dynamic changes in data will play a key role
- Infrastructure Challenge
 - Applications will have to spread and run across different heterogeneous infrastructures (possibly even simultaneously)
 - Probably other objectives besides "minimise makespan"

What are the Upcoming Challenges ?



ole.weidner@ed.ac.uk

Mar 18, 2011

What are the Upcoming Challenges ?



• The Data Deluge

- "The amount of genomic data available for study is increasing at a rate similar to that of Moore's Law" http://www.mcs.anl.gov/uploads/cels/papers/P1238.pdf
- "Store now process later" might not always be an option
- Adaptive algorithms will be required to handle vast amounts of dynamic (streaming) data
- Methods to characterise/predict data and especially dynamic changes in data will play a key role
- Infrastructure Challenge
 - Applications will have to spread and run across different heterogeneous infrastructures (possibly even simultaneously)
 - Probably other objectives besides "minimise makespan"

Can Distributed Abstractions Help ?



- There might be lots of new challenges, but from an application perspective things will mostly stay the same (!)
 - Applications will still be using common patterns and abstractions
 - But: input and objectives might change and they might change DYNAMICALLY
- Once we have understood the details and dynamics of these new challenges, we can *encapsulate* them inside distributed programming abstractions
- If this hasn't been done, it could be a nice intellectual and practically relevant contribution to the field

A Case for Dynamic Optimisation

O-O-O edinburgh O-O data-intensive O-O-O research

- DPAs supporting **dynamic** optimisation to address dynamic data challenges:
 - Adaptive execution strategy based on input, output and system characteristics
 - Ability to "understand" and "predict" data (characteristics)
- DPAs supporting **autonomous** adaption to address rising complexity in infrastructure



- There doesn't seem to be a lot of work that revisits DPAs in the context of:
 - Streaming input / output data
 - Data with dynamically changing characteristics
 - Autonomic | adaptive | dynamic | optimisation
- Lots of work in static optimisation ("fine-tuning")
 - Optimise throughput for workload X on (idealised) platform Y
 - Usually confined to a specific middleware
- There seems to be a lot of uncharted territory, but that remains to be proved (more reading!)

Current State of Affairs



IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 21, NO. 1, JANUARY 2010

All-Pairs: An Abstraction for Data-Intensive Computing on Campus Grids

Christopher Moretti, *Student Member, IEEE*, Hoang Bui, *Student Member, IEEE*, Karen Hollingsworth, Brandon Rich, Patrick Flynn, *Senior Member, IEEE*, and Douglas Thain, *Member, IEEE*

Abstract—Today, campus grids provide users with easy access to thousands of CPUs. However, it is not always easy for nonexpert users to harness these systems effectively. A large workload composed in what seems to be the obvious way by a naive user may accidentally abuse shared resources and achieve very poor performance. To address this problem, we argue that campus grids should provide end users with high-level abstractions that allow for the easy expression and efficient execution of data-intensive workloads. We present one example of an abstraction—All-Pairs—that fits the needs of several applications in biometrics, bioinformatics, and data mining. We demonstrate that an optimized All-Pairs abstraction is both easier to use than the underlying system, achieve performance orders of magnitude better than the obvious but naive approach, and is both faster and more efficient than a tuned conventional approach. This abstraction has been in production use for one year on a 500 CPU campus grid at the University of Notre Dame and has been used to carry out a groundbreaking analysis of biometric data.

Index Terms—All-pairs, biometrics, cloud computing, data intensive computing, grid computing.

1 INTRODUCTION

MANY fields of science and engineering have the potential to use large numbers of CPUs to attack problems of enormous scale. Campus-scale computing grids are now a standard tool employed by many academic institutions to provide large-scale computing power. Using middleware such as Condor [40] or Globus [21], many disparate clusters and stand-alone machines may be joined into a single computing system with many providers and consumers. Today, campus grids of about one thousand machines are commonplace [41], and are being grouped into larger structures, such as the 20,000-CPU Indiana Diagrid and the 40,000-CPU Open Science Grid [36].

Campus grids have the unique property that consumers of the system must always defer to the needs of the resource providers. For example, if a desktop computer is donated to the campus grid, then a visiting job may use it during idle times, but will be preempted when the owner is busy at the keyboard. If a research cluster is donated to the campus grid, visiting jobs may use it, but might be preempted by higher priority batch jobs submitted by the owner of the cluster. In short, the user of the system has access to an enormous number of CPUs, but must expect to be preempted from many of them as a normal condition.

Because of this property, scaling up an application to a campus grid is a nontrivial undertaking. Parallel libraries and languages such as MPI [18], OpenMP [14], and Cilk [8] are not usable in this context because they do not explicitly address preemption and failure as a normal case. Instead,

 The authors are with the Department of Computer Science and Engineering, University of Notre Dame, 384 Fitzpatrick Hall, Notre Dame, IN 46556.
 E-mail: (cmoretti, hbui, kholling, brich, flynn, dthain)@nd.edu.

Manuscript received 22 July 2008; revised 11 Feb. 2009; accepted 9 Mar. 2009; published online 13 Mar. 2009. Recommended for acceptance by D. Bader.

For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2008-07-0277. Digital Object Identifier no. 10.1109/TPDS.2009.49.

1045-9219/10/\$26.00 © 2010 IEEE Publishe

large workloads must be specified as a set of sequential processes connected by files. End users must carefully arrange the I/O behavior of their workloads. Bad configurations can result in poor performance, outright failure of the application, and abuse of physical resources shared by others. All too often, an end user composes a workload that runs correctly on one machine, then on 10 machines, but fails disastrously on 1,000 machines.

Providing an *abstraction* is one approach to avoiding these problems. An abstraction allows a user to declare a workload composed of multiple sequential programs and the data that they process, while hiding the details of how the workload will be realized in the system. Abstracting away details hides complications that are not apparent or important to a novice, limiting the opportunity for disasters. Because an abstraction states a workload in a declarative way, it can be realized within the grid in whatever way satisfies cost, policy, and performance constraints. Abstractions could also be implemented in other kinds of systems, such as dedicated clusters or multicore CPUs, but we do not address those here.

We have implemented one such abstraction—All-Pairs —for a class of problems found in many fields. All-Pairs is the Cartesian product of a large number of objects with a custom comparison function. While simple to state, it is nontrivial to carry out on large problems that require hundreds of nodes running for several days. All-Pairs is similar in spirit to other abstractions such as Dryad [28], Map-Reduce [16], Pegasus [17], and Swift [42], but it addresses a different category of applications.

Our implementation of All-Pairs is currently in production use on a 500 CPU campus grid at the University of Notre Dame, using Condor [40] to manage the CPUs and Chirp [39] to manage the storage. To demonstrate the performance benefits of using an abstraction, we compare two different implementations. The *conventional* implementation executes the specification by simply submitting a series of batch jobs that use a central file server to read data on demand and write Published to the IEEE Computer Society

be a lot of work that revisits

ut data changing characteristics dynamic | optimisation ptimisation ("fine-tuning") or workload X on (idealised) platform Y becific middleware t of uncharted territory, but that

more reading!)

Current State of Affairs

large workloads mus

processes connected

urations can result in

the application, and a

others. All too often, a

runs correctly on one

fails disastrously on

Providing an abst

these problems. An al

workload composed of

the data that they proc

the workload will be a

away details hides con

important to a novice, li

Because an abstraction

way, it can be realized

satisfies cost, policy, ar

tions could also be imr

such as dedicated clust

We have impleme

-for a class of prob

is the Cartesian prod

a custom comparison

nontrivial to carry

hundreds of nodes

similar in spirit to

Map-Reduce [16],

addresses a differen

tion use on a 500 CPU

Dame, using Condor

to manage the stora

benefits of using an

implementations. The

the specification by s that use a central file s

Our implementation

address those here.

arrange the I/O beha



IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 21, NO. 1, JANUARY 2010

All-Pairs: An Abstraction for Da Computing on Campus (

Christopher Moretti, *Student Member, IEEE*, Hoang Bui, *Student Membe* Brandon Rich, Patrick Flynn, *Senior Member, IEEE*, and Dougla

Abstract—Today, campus grids provide users with easy access to thousands of CPUs. Hower users to harness these systems effectively. A large workload composed in what seems to be a accidentally abuse shared resources and achieve very poor performance. To address this proble provide end users with high-level abstractions that allow for the easy expression and efficient e We present one example of an abstraction—All-Pairs—that fits the needs of several applicatio data mining. We demonstrate that an optimized All-Pairs abstraction is both easier to use than performance orders of magnitude better than the obvious but naive approach, and is both fast conventional approach. This abstraction has been in production use for one year on a 500 CPU Dame and has been used to carry out a groundbreaking analysis of biometric data.

Index Terms-All-pairs, biometrics, cloud computing, data intensive computing, grid computi

1 INTRODUCTION

MANY fields of science and engineering have the potential to use large numbers of CPUs to attack problems of enormous scale. Campus-scale computing grids are now a standard tool employed by many academic institutions to provide large-scale computing power. Using middleware such as Condor [40] or Globus [21], many disparate clusters and stand-alone machines may be joined into a single computing system with many providers and consumers. Today, campus grids of about one thousand machines are commonplace [41], and are being grouped into larger structures, such as the 20,000-CPU Indiana Diagrid and the 40,000-CPU Open Science Grid [36].

Campus grids have the unique property that consumers of the system must always defer to the needs of the resource providers. For example, if a desktop computer is donated to the campus grid, then a visiting job may use it during idle times, but will be preempted when the owner is busy at the keyboard. If a research cluster is donated to the campus grid, visiting jobs may use it, but might be preempted by higher priority batch jobs submitted by the owner of the cluster. In short, the user of the system has access to an enormous number of CPUs, but must expect to be preempted from many of them as a normal condition.

Because of this property, scaling up an application to a campus grid is a nontrivial undertaking. Parallel libraries and languages such as MPI [18], OpenMP [14], and Cilk [8] are not usable in this context because they do not explicitly address preemption and failure as a normal case. Instead,

 The authors are with the Department of Computer Science and Engineering, University of Notre Dame, 384 Fitzpatrick Hall, Notre Dame, IN 46556.
 E-mail: [cmoretic, hbui, kholling, brich, flynn, dthain]@nd.edu.

Manuscript received 22 July 2008; revised 11 Feb. 2009; accepted 9 Mar. 2009; published online 13 Mar. 2009. Recommended for acceptance by D. Bader.

For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2008-07-0277. Digital Object Identifier no. 10.1109/TPDS.2009.49.

1045-9219/10/\$26.00 © 2010 IEEE Published by the IEEE Comp

Cluster Comput (2010) 13: 243-256 DOI 10.1007/s10586-010-0134-7

Harnessing parallelism in multicore clusters with the All-Pairs, Wavefront, and Makeflow abstractions

Li Yu · Christopher Moretti · Andrew Thrasher · Scott Emrich · Kenneth Judd · Douglas Thain

Received: 9 November 2009 / Accepted: 16 March 2010 / Published online: 23 April 2010 © Springer Science+Business Media, LLC 2010

Abstract Both distributed systems and multicore systems are difficult programming environments. Although the expert programmer may be able to carefully tune these systems to achieve high performance, the non-expert may struggle. We argue that high level abstractions are an effective way of making parallel computing accessible to the non-expert. An abstraction is a regularly structured framework into which a user may plug in simple sequential programs to create very large parallel programs. By virtue of a regular structure and declarative specification, abstractions may be materialized on distributed, multicore, and distributed multicore systems with robust performance across a wide range of problem sizes. In previous work, we presented the All-Pairs abstraction for computing on distributed systems of single CPUs. In this paper, we extend All-Pairs to multicore systems, and introduce the Wavefront and Makeflow abstractions, which represent a number of problems in economics and bioinformatics. We demonstrate good scaling of both abstractions up to 32 cores on one machine and hundreds of cores in a distributed system

Keywords Abstractions · Multicore · Distributed systems · Bioinformatics · Economics

L. Yu (⊠) · C. Moretti · A. Thrasher · S. Emrich · D. Thain Department of Computer Science and Engineering, University of Norte Dame, South Bend, USA e-mail: lyu2@nd.edu

K. Judd Hoover Institution, Stanford University, Stanford, USA

1 Introduction

Distributed systems such as clusters, clouds, and grids are very challenging programming environments. (Hereafter, we refer to all of these systems as *clusters*.) A user that wishes to execute a large workload with some inherent parallelism is confronted with a dizzying array of choices. How should the workload be broken up into jobs? How should the data be distributed to each node? How many nodes should be used? Will the network be a bottleneck? Often, the answers to these questions depend heavily on the properties of the system and workload in use. Changing one parameter, such as the size of a file or the runtime of a job, may require a completely different strategy.

Multicore systems present many of the same challenges. The orders of magnitude change, but the questions are similar. How should work be divided among threads? Should we use message passing or shared memory? How many CPUs should be used? Will memory access present a bottleneck? When we consider clusters of multicore computers, then the problems become more complex.

We argue that *abstractions* are an effective way of enabling non-expert users to harness clusters, multicore computers, and clusters of multicore computers. An abstraction is a declarative structure that joins simple data structures and small sequential programs into parallel graphs that can be scaled to very large sizes. Because an abstraction is specialized to a restricted class of workloads, it is possible to create an efficient, robust, scalable, and fault tolerant implementation. In previous work, we introduced the All-Pairs [12] and Classify [13] abstractions, and described how they can be used to solve data intensive problems in the fields of biometrics, bioinformatics, and data mining. Our implementations allow non-experts to harness hundreds of processors on problems that run for hours or days using the Condor [27] distributed batch system.

D Springer

nat revisits

on uning")

CS

ealised) platform Y

ritory, but that

19

Current State of Affairs

large workloads mus

processes connected

urations can result in

the application, and a

others. All too often, a

runs correctly on one

fails disastrously on

Providing an abstr

these problems. An al

workload composed of

the data that they proc

the workload will be a

away details hides con

important to a novice, li

Because an abstraction

way, it can be realized

satisfies cost, policy, ar

tions could also be imr

such as dedicated clust

We have impleme

-for a class of prob

is the Cartesian prod

a custom comparison

nontrivial to carry

hundreds of nodes

similar in spirit to

Map-Reduce [16],

addresses a differen

tion use on a 500 CPU

Dame, using Condor

to manage the stora

benefits of using an

implementations. The

the specification by s that use a central file s

Our implementation

address those here.

arrange the I/O beha



IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 21, NO. 1, JANUARY 2010

All-Pairs: An Abstraction for Da Computing on Campus (

Christopher Moretti, *Student Member, IEEE*, Hoang Bui, *Student Membe* Brandon Rich, Patrick Flynn, *Senior Member, IEEE*, and Dougla

Abstract—Today, campus grids provide users with easy access to thousands of CPUs. Hower users to harness these systems effectively. A large workload composed in what seems to be accidentally abuse shared resources and achieve very poor performance. To address this proble provide end users with high-level abstractions that allow for the easy expression and efficient e We present one example of an abstraction—All-Pairs—that fits the needs of several application data mining. We demonstrate that an optimized All-Pairs abstraction is both easier to use than performance orders of magnitude better than the obvious but naive approach, and is both fast conventional approach. This abstraction has been in production use for one year on a 500 CPU Dame and has been used to carry out a groundbreaking analysis of biometric data.

Index Terms-All-pairs, biometrics, cloud computing, data intensive computing, grid computin

1 INTRODUCTION

MANY fields of science and engineering have the potential to use large numbers of CPUs to attack problems of enormous scale. Campus-scale computing grids are now a standard tool employed by many academic institutions to provide large-scale computing power. Using middleware such as Condor [40] or Globus [21], many disparate clusters and stand-alone machines may be joined into a single computing system with many providers and consumers. Today, campus grids of about one thousand machines are commonplace [41], and are being grouped into larger structures, such as the 20,000-CPU Indiana Diagrid and the 40,000-CPU Open Science Grid [36].

Campus grids have the unique property that consumers of the system must always defer to the needs of the resource providers. For example, if a desktop computer is donated to the campus grid, then a visiting job may use it during idle times, but will be preempted when the owner is busy at the keyboard. If a research cluster is donated to the campus grid, visiting jobs may use it, but might be preempted by higher priority batch jobs submitted by the owner of the cluster. In short, the user of the system has access to an enormous number of CPUs, but must expect to be preempted from many of them as a normal condition.

Because of this property, scaling up an application to a campus grid is a nontrivial undertaking. Parallel libraries and languages such as MPI [18], OpenMP [14], and Cilk [8] are not usable in this context because they do not explicitly address preemption and failure as a normal case. Instead,

 The authors are with the Department of Computer Science and Engineering, University of Notre Dame, 384 Fitzpatrick Hall, Notre Dame, IN 46556.
 E-mail: [cmoretti, hbui, kholling, brich, flynn, dthain/@nd.edu.

Manuscript received 22 July 2008; revised 11 Feb. 2009; accepted 9 Mar. 2009; published online 13 Mar. 2009. Recommended for acceptance by D. Bader.

For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2008-07-0277. Digital Object Identifier no. 10.1109/TPDS.2009.49.

1045-9219/10/\$26.00 © 2010 IEEE Published by the IEEE Comp

Cluster Comput (2010) 13: 243–256 DOI 10.1007/s10586-010-0134-7

Harnessing parallelism in multicore clusters with t Wavefront, and Makeflow abstractions

Li Yu · Christopher Moretti · Andrew Thrasher · Scott Emrich · Kenneth Judd · Douglas Thain

Received: 9 November 2009 / Accepted: 16 March 2010 / Published online: 23 April 2010 © Springer Science+Business Media, LLC 2010

Abstract Both distributed systems and multicore systems are difficult programming environments. Although the expert programmer may be able to carefully tune these systems to achieve high performance, the non-expert may struggle. We argue that high level abstractions are an effective way of making parallel computing accessible to the non-expert. An abstraction is a regularly structured framework into which a user may plug in simple sequential programs to create very large parallel programs. By virtue of a regular structure and declarative specification, abstractions may be materialized on distributed, multicore, and distributed multicore systems with robust performance across a wide range of problem sizes. In previous work, we presented the All-Pairs abstraction for computing on distributed systems of single CPUs. In this paper, we extend All-Pairs to multicore systems, and introduce the Wavefront and Makeflow abstractions, which represent a number of problems in economics and bioinformatics. We demonstrate good scaling of both abstractions up to 32 cores on one machine and hundreds of cores in a distributed system

Keywords Abstractions · Multicore · Distributed systems · Bioinformatics · Economics

L. Yu (⊠) · C. Moretti · A. Thrasher · S. Emrich · D. Thain Department of Computer Science and Engineering, University of Notre Dame, South Bend, USA e-mail: lyu2@nd.edu K. Judd Hoover Institution. Stanford University. Stanford, USA

Distributed systems very challenging pi we refer to all of wishes to execute a allelism is confronte should the workload data be distributed be used? Will the n swers to these quest the system and wor such as the size of a 1 a completely differen Multicore systems The orders of magnit lar. How should work use message passing should be used? Will When we consider ch problems become mc We argue that al abling non-expert us puters, and clusters

1 Introduction

puters, and clusters is a declarative struct small sequential proscaled to very largeized to a restricted an efficient, robust, tion. In previous wo Classify [13] abstraused to solve data metrics, bioinforma

Classify [13] abstraused to solve data i metrics, bioinforma tions allow non-exp problems that run 1 distributed batch sy All-Pairs: An Abstraction for Data-Intensive Cloud Computing

Christopher Moretti, Jared Bulosan, Douglas Thain, and Patrick J. Flynn Department of Computer Science and Engineering, University of Notre Dame *

Abstract

Although modern parallel and distributed computing systems provide easy access to large amounts of computing power, it is not always easy for non-expert users to harness these large systems effectively. A large workload composed in what seems to be the obvious way by a naive user may accidentally abuse shared resources and achieve very poor performance. To address this problem, we propose that production systems should provide end users with high-level abstractions that allow for the easy expression and efficient execution of data intensive workloads. We present one example of an abstraction - All-Pairs - that fits the needs of several data-intensive scientific applications. We demonstrate that an optimized All-Pairs abstraction is both easier to use than the underlying system, and achieves performance orders of magnitude better than the obvious but naive approach, and twice as fast as a hand-optimized conventional approach.

1 Introduction

Many scientists have large problems that can make use of distributed computing; however, most also are not distributed computing experts. Without distributed computing experience and expertise, it is difficult to navigate the large number of factors involved in large distributed systems and the software that harnesses these resources. Inadvertent poor choices can result in poor performance or even outright failures of the application. Poor choices can also lead to inefficient use of shared resources and abuse of the distributed system's infrastructure such as job queues and matchmaking software.

Providing an abstraction useful for a class of problems is one approach to avoiding the pitfalls of distributed computing. Such an abstraction gives the user an interface to define their problem in terms of data and computation requirements, while hiding the details of how the problem will be realized in the system. Abstracting away details also hides

*This work was supported in part by National Science Foundation grants CCF-06-21434 and CNS-06-43229.

978-1-4244-1694-3/08/\$25.00 ©2008 IEEE

complications that are not apparent to a novice, limiting the opportunity for disastrous decisions that result in pathological cases. The goal is not to strip power from smart users, but rather to make distributed computing accessible to nonexperts.

We have implemented one such abstraction – All-Pairs – for a class of problems found in several scientific fields. This implementation has several broad steps. First, we model the workflow so that we may predict execution based on grid and workload parameters, such as the number of hosts. We distribute the data to the compute nodes via a spanning tree, choosing sources and targets in a flexible manner. We dispatch batch jobs that are structured to provide good results based on the model. Once the batch jobs have completed, we collect the results into a canonical form for the end-user, and delete the scratch data left on the compute nodes.

We also examine two algorithms for serving the workload's data requirement: demand paging similar to a traditional cluster and active storage. Active storage delivers higher throughput and efficiency for several large workloads on a shared distributed system, and can result in total workload turnaround times that are up to twice as fast.

We evaluate the abstraction's model, execution, and data delivery on All-Pairs problems in biometrics and data mining on a 500-CPU shared computing system. We have found turnaround time with the abstraction is orders of magnitude faster than for workloads configured using nonexperts' choices.

2 The All-Pairs Problem

The All-Pairs problem is easily stated:

All-Pairs(set A, set B, function F) returns matrix M: Compare all elements of set A to all elements of set B via function F, yielding matrix M, such that M[i,j] = F(A[i],B[j]).

Variations of the All-Pairs problem occur in many branches of science and engineering, where the goal is either to understand the behavior of a newly created function

Mar 18,2011

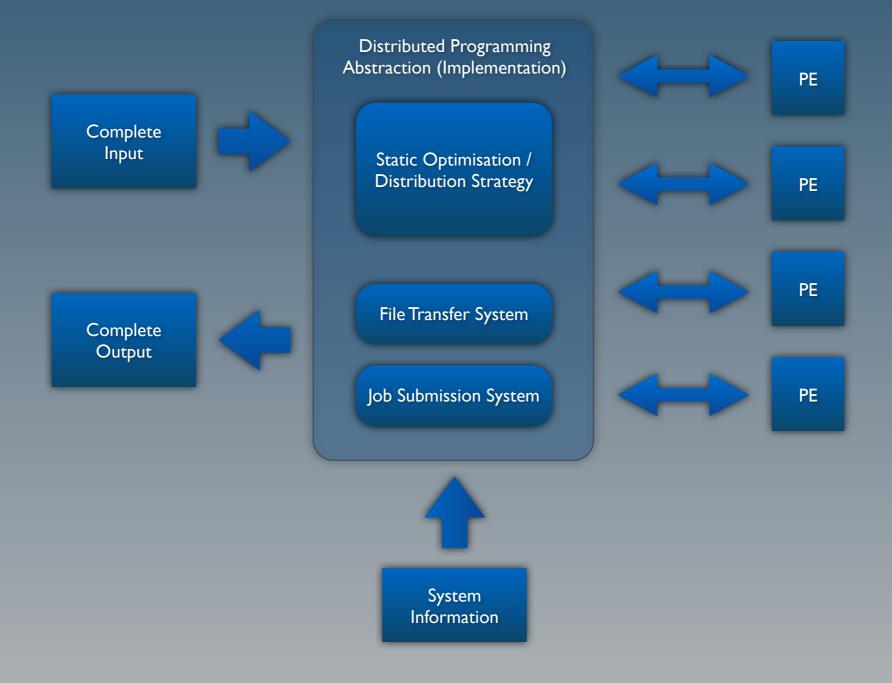
ole.weidner@ed.ac.uk

Friday, March 18, 2011

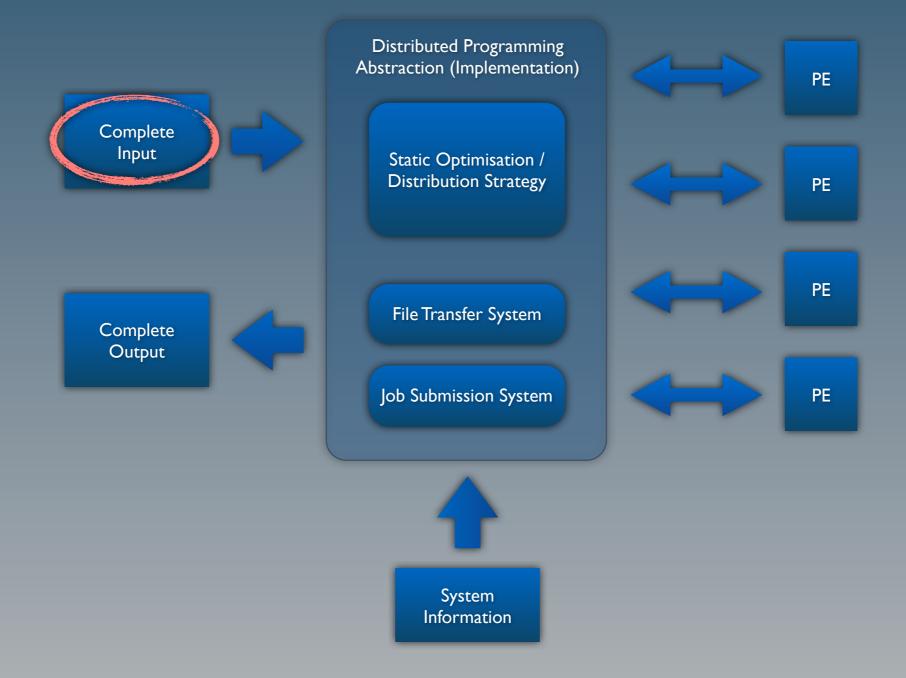


- There doesn't seem to be a lot of work that revisits DPAs in the context of:
 - Streaming input / output data
 - Data with dynamically changing characteristics
 - Autonomic | adaptive | dynamic | optimisation
- Lots of work in static optimisation ("fine-tuning")
 - Optimise throughput for workload X on (idealised) platform Y
 - Usually confined to a specific middleware
- There seems to be a lot of uncharted territory, but that remains to be proved (more reading!)

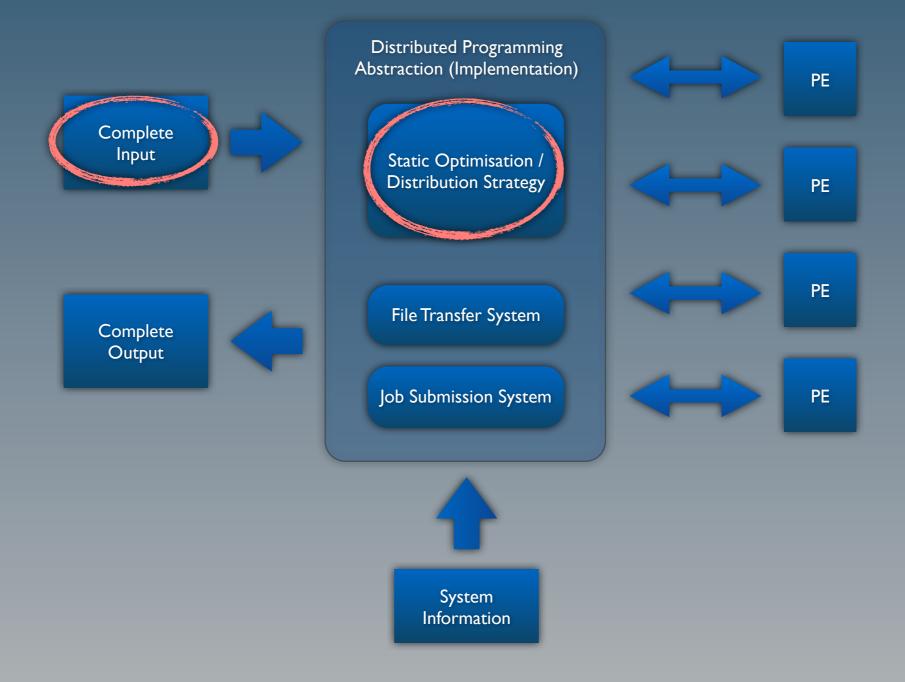




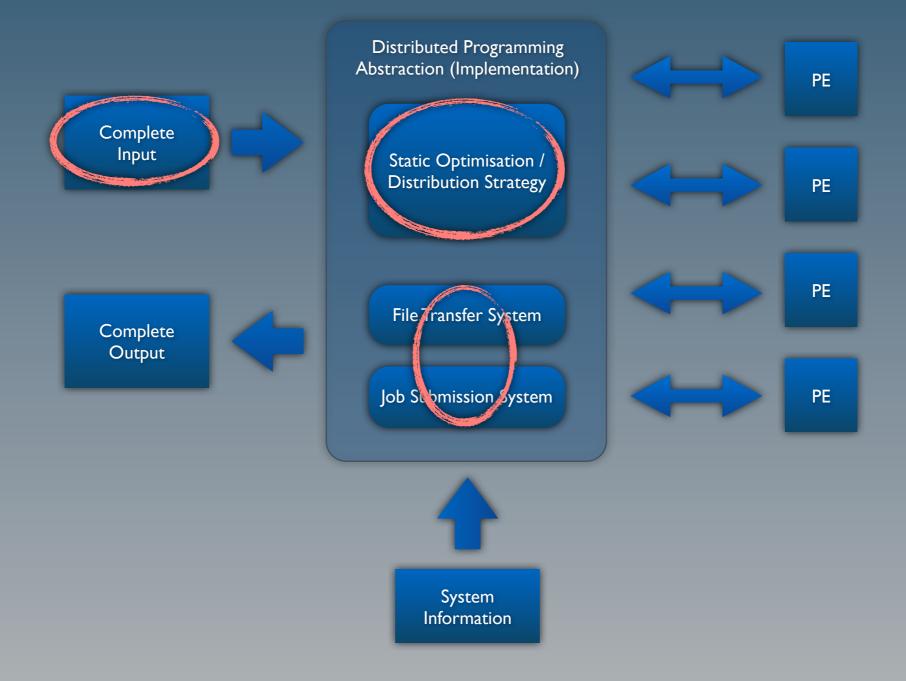




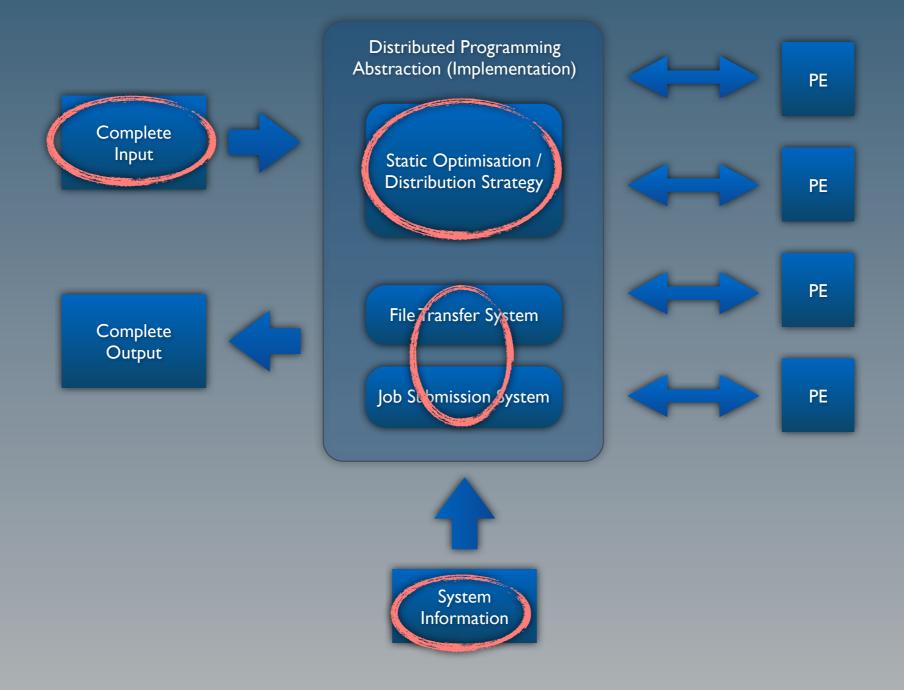




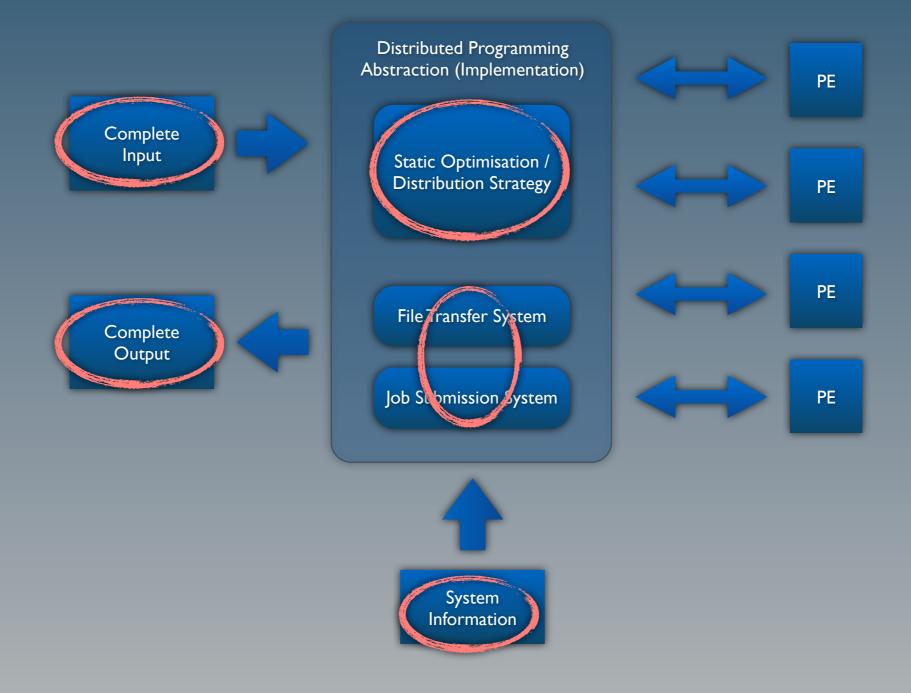












A Somewhat Concrete Plan

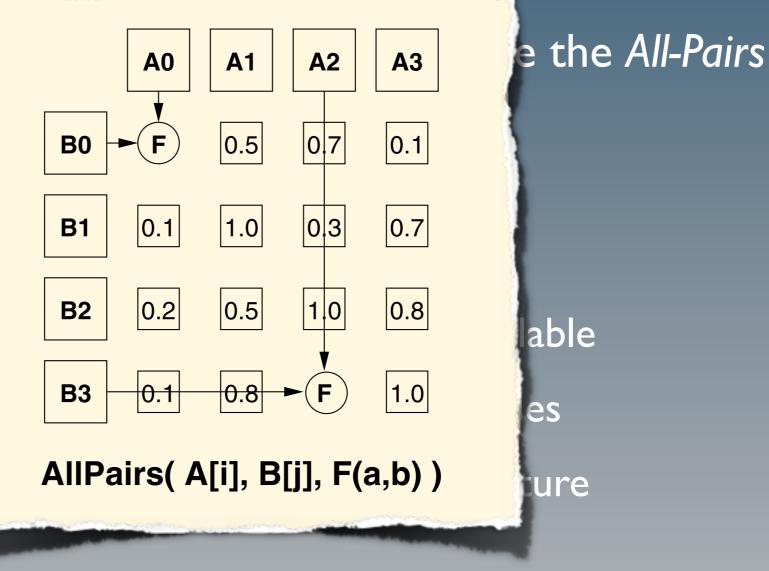


- Pick a common distributed abstraction and make it a "Dynamically Optimising and Adaptive Abstraction" (working title)
- A really good candidate so far seems to be the All-Pairs abstraction:
 - Simple and generic
 - Well defined input and output
 - Lots of real-world applications and data available
 - Many potential dynamic and big-data use-cases
 - Not confined to a specific type of infrastructure

A Somewhat Concrete Plan

O-O-O edinburgh O-O data-intensive

- Pick a common distributed abstraction and make it a "Dynamically Optimising and Adaptive Abstraction" (working title)
- A really good abstraction:
 - Simple and
 - Well define
 - Lots of real
 - Many poter
 - Not confine



A Somewhat Concrete Plan



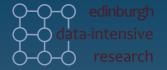
- Pick a common distributed abstraction and make it a "Dynamically Optimising and Adaptive Abstraction" (working title)
- A really good candidate so far seems to be the All-Pairs abstraction:
 - Simple and generic
 - Well defined input and output
 - Lots of real-world applications and data available
 - Many potential dynamic and big-data use-cases
 - Not confined to a specific type of infrastructure



• Bioinformatics

- Phylogenetic tree generation
- Sequence Alignment
- Biometrics
 - Feature (e.g., face) detection
- Machine Learning
 - Unsupervised learning (e.g., clustering, density estimation)
 - Evaluation of new learning functions
- *N*-body simulation (Cosmology)

All-Pairs Application Areas (cont.)



• Web / Data-mining

- (Near) duplicate document detection
 - "more-like-this queries", collaborative filtering
 - "data-shedding", duplicate deletion
- Query refinement
- Coalition detection
- Interesting: some of the matching algorithms in data-mining already support speed v.s. accuracy tuning

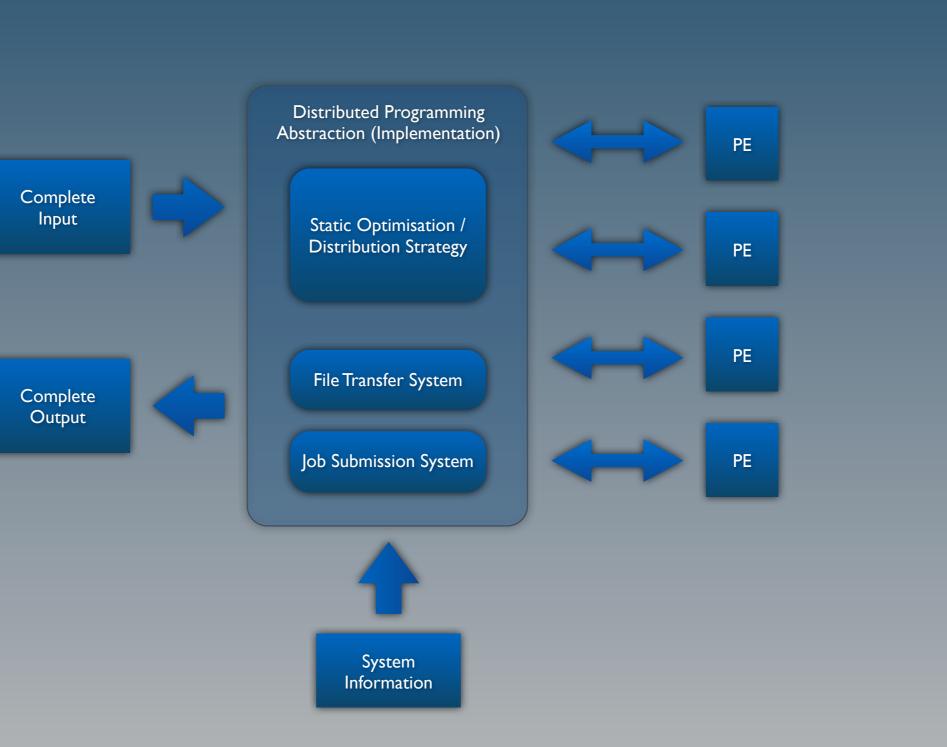


- Develop a model that helps us to understand the relevant aspects of data in an All-Pairs context
- Define the characteristics we need to capture / extract in order to support dynamic decision-making
 - Static / dynamic ?
 - Rate ?
 - Dependency ?
 - Affinity / Relationship ?

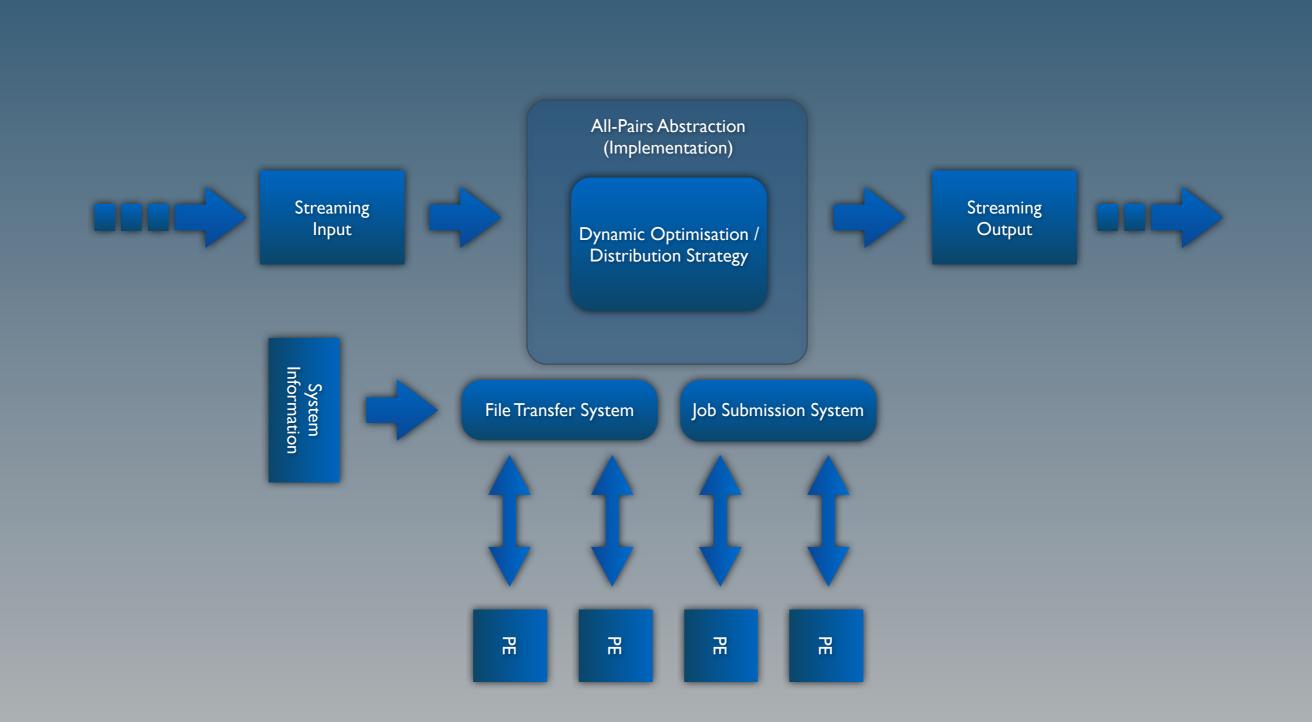


Revisit the issue of optimisation objectives

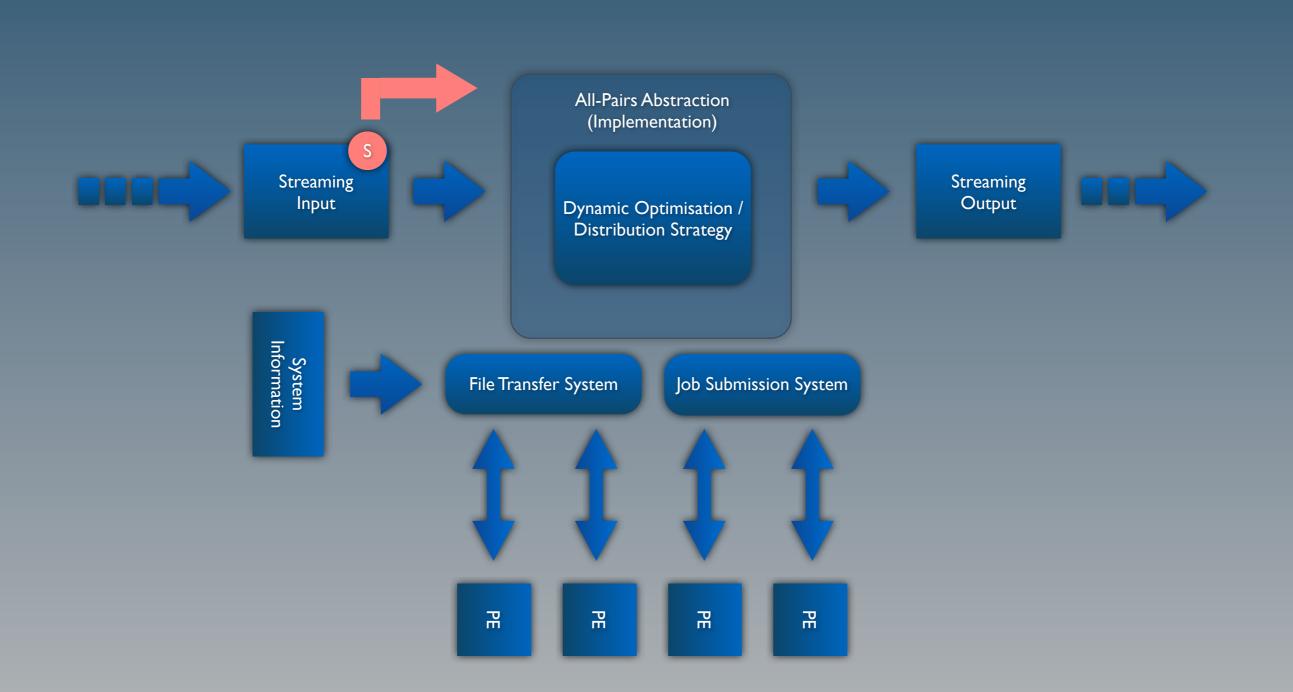
- Minimise energy consumption ?
- Get me the best possible solution within X hours ?
- Don't exceed bandwidth X or storage Y ?
- Develop an optimisation framework
 - Possibly based on control theory concepts
 - "Control plant" being the All-Pairs execution framework
 - "Sensors" measuring input and output data
 - "Controller" objective-specific optimisation rules and functions

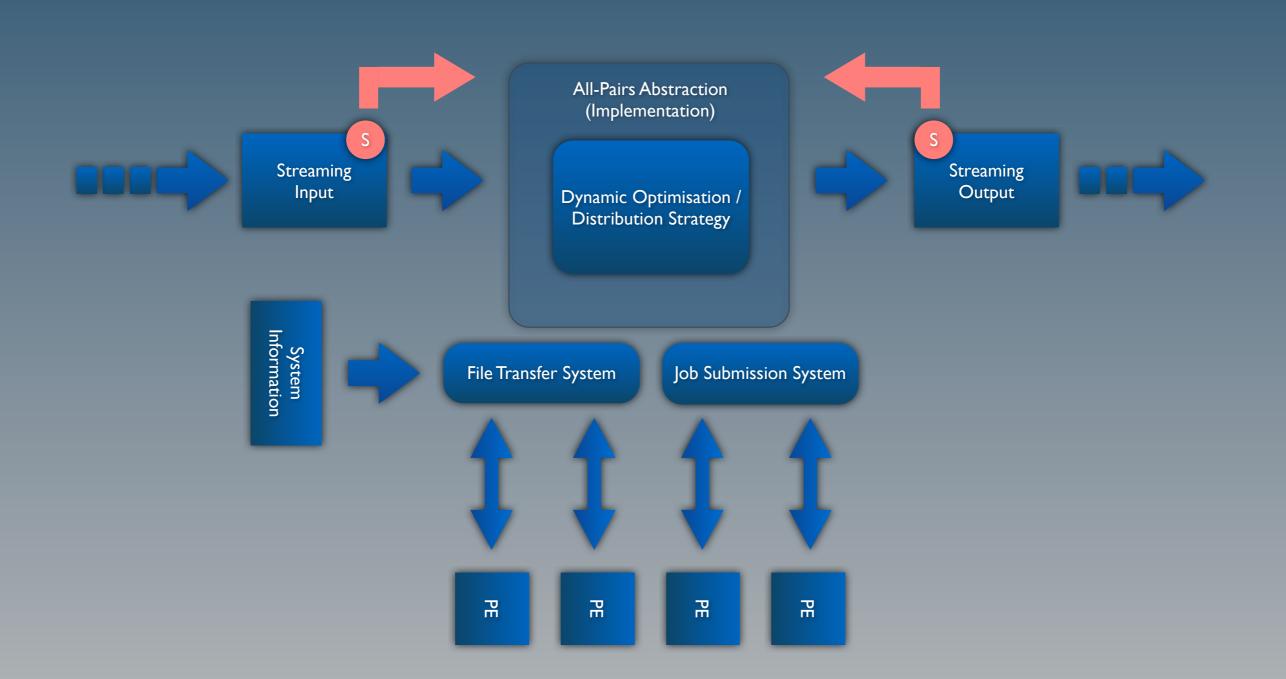






Friday, March 18, 2011





27

Lots of Open Questions



- Can we predict data ?
- How do the optimisation rules and functions look like ?
 - Finite set of rules ?
 - Machine Learning ?
 - Bayesian Networks ?
 - ...
 - A combination of the above ?
- Is dynamic optimisation cheap enough to use it with static problems as well ?



- What is the right level of abstraction ?
- How do we implement sensors ?
- How do we implement actuators ?
- How would an unobtrusive 'user-interface' look like ?
- Can we be completely infrastructure independent ?
- What about fault-tolerance ?
- Can we develop a generic dynamic optimisation framework that can be used not just for All-Pairs ?



• How can we find real-world dynamic applications ?

- "Social engineering"
- Can we generate synthetic workload ?
- What are the testbeds we should use ?
 - TeraGrid (HPC Grid)
 - EGI (HTC Grid)
 - Clouds ?



SAGA:

C++/Python:	<u>http://saga.cct.lsu.edu</u>
OGF GFD.90:	http://www.ogf.org/documents/GFD.90.pdf
Mailing-Lists:	<u>saga-users@cct.lsu.edu</u>

My Ph.D. Research:

Homepage:	http://www.oleweidner.com
Code Repo.:	https://github.com/oweidner
Email:	<u>ole.weidner@ed.ac.uk</u>