Introduction to Grids Tutorial

SuperComputing '07

People:

Ben Clifford – developer with the University of Chicago Computation Institute – works on grid workflow development (which you'll see at the very end of this talk) and on the Open Science Grid education programme (which is what this tutorial is heavily based on).

Alina Bejan – the OSG Education Coordinator (EOT) who works at the CI;

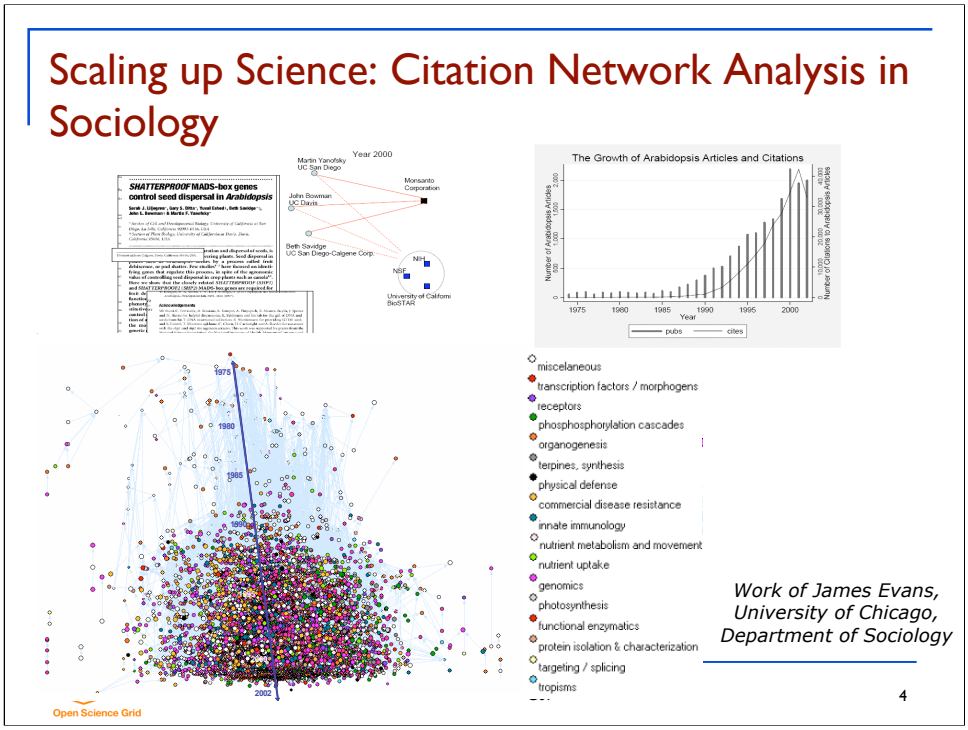
Mike Wilde -- also works on the Education program OSG EOT, on Swift and on many other projects

Roadmap

- Motivation
- What is the grid?
- How do we work with a grid?
- What's next?

Motivation

- Example



Our example problem is the Citation Analysis

Citation Analysis is the most common method of bibliometrics. Citation analysis uses citations in scholarly works to establish links to other works or other researchers. Co-citation coupling and bibliographic coupling are specific kinds of citation analysis.

Citation analysis is the examination of the frequency and pattern of citations in articles and books.

Due to unprecedented growth of electronic resource (e-resource) availability, one of the questions currently being explored is, "how often are e-resources being cited in my field?"

The graph that represents the 'reference' relationships is not easy to compute

Picture: citation network analysis for the case of plant biotechnology; the relationships among scientists and the public, private, and nonprofit organizations involved in research on the genetic model organism Arabidopsis thaliana are represented

Scaling up the analysis

- Query and analysis of 25+ million citations
- Work started on desktop workstations
- Queries grew to month-long duration
- With data distributed across U of Chicago TeraPort **cluster**
  - Advantages:
    - 50 (faster) CPUs gave 100 X speedup
    - Many more methods and hypotheses can be tested!
- Higher *throughput* and *capacity* enables *deeper analysis* and *broader community access*.

Open Science Grid

Intro to Grids / SC07 5

This project started running on a desktop workstation, as most projects do – you write the code on your PC and it works.

But as you start feeding in more data, it starts taking longer and longer to process – it started off taking a few minutes to run with a small number of citations but as his citation database got bigger the program slowed down.

Eventually he had over 25 million citations and the program was taking a month to run on his PC – every time he made a change and wanted to try it out, it took a month to process everything.

So one thing we did was to move the problem onto UC's grid site, the Teraport – using 50 CPUs instead of his one PC, and faster CPUs at that, he got 100x speedup – go from a month back down to running his jobs in hours.

This way, he can get his results faster and he can do much deeper analysis than he could with one PC.

It is possible to expand further, using even more available computing and storage capacity for this application.

A desktop workstation is yours to do what you like.

- ~2 GHz CPU
- ~500 GB
- ~30 Gigaflops

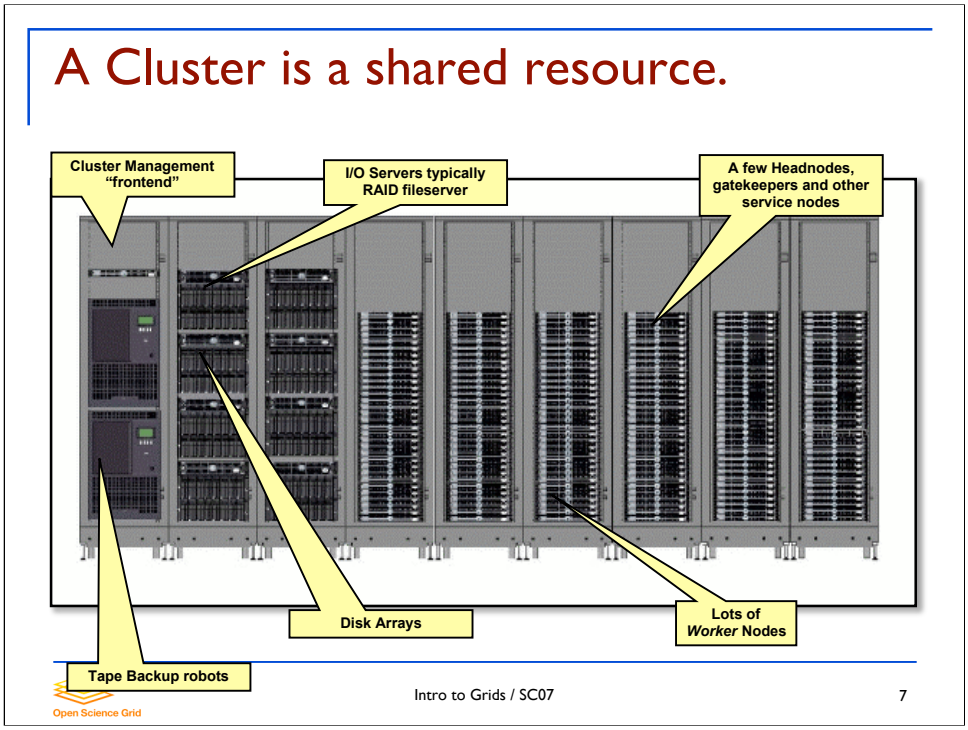
Open Science Grid

Intro to Grids / SC07 6

Here is the scaling up problem again:

We start with a PC – CPU around 2Ghz, storage of maybe half a terabyte.

You own it yourself – yours to use and abuse.



Your department might purchase and manage a cluster– illustrated above.

A cluster has many "compute nodes", (plenty of) storage, and network connectivity. (like putting multiple computers to work together)

Basically it provides a shared computing resource for tasks that are too demanding for your PC, where you can't afford to have your own dedicated big computer.

A regular user (you) will probably use it only part of the time, so it makes sense to share it.

It is possible that everyone in your department gets access to this cluster.

Grids represent a different approach

– Build bigger supercomputers by joining smaller ones together in a ***grid***

Open Science Grid

The OSG

Origins:
- National Grid (iVDGL, GriPhyN, PPDG) and LHC Software & Computing Projects

Current Compute Resources:
- 61 Open Science Grid sites
- Connected via Inet2, NLR.... from 10 Gbps – 622 Mbps
- Compute & Storage Elements
- All are Linux clusters
- Most are shared
  - Campus grids
  - Local non-grid users
- More than 10,000 CPUs
  - A lot of opportunistic usage
  - Total computing capacity difficult to estimate
  - Same with Storage

Open Science Grid

Intro to Grids / SC07 8

We are now ready to introduce the notion of a grid, and grid computing.

There exists a different approach to the cluster we just saw, which is the grid approach.

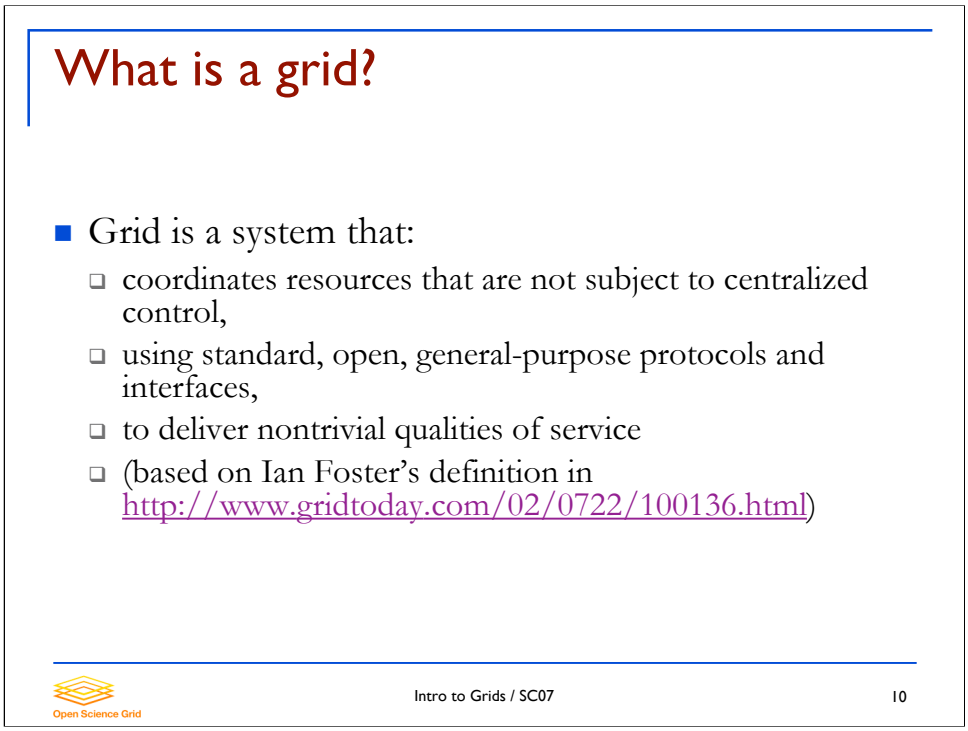
instead of building a bigger and bigger single cluster, you federate together a lot of smaller ones.

For example, the OSG has 60 sites used by many thousands of users spread mostly around the United States but also in other countries.

PC vs Cluster vs Grid

- PC:
  - Owner has total control
  - Limited capabilities
- Cluster:
  - Used by a small number of people using (e.g., department, institution)
  - – Preserves some locality
- Grid:
  - Thousands of users - large scale
  - From many different places - highly distributed
  - Increased problems (due to distributivity aspect)

As we transition from PC up to a grid, though, many things change.



Different definitions.

Let's consider Ian Foster's three point checklist – a grid is an infrastructure that brings resources together without imposing central control. Computers, storage space, network connections, databases are examples of resources.

Why are open protocols important? So that you can use different software – you don't have to buy special programs to use it and be bound by certain licensing limitations. Offers freedom and flexibility in working on your own terms, allowing you (the grid user) to make decisions about your local infrastructure.

A major advantage is that a grid is capable of delivering non-trivial qualities of service – basically it means that it doesn't just break at the first opportunity. Reliability is important especially for large scale problems as those that are usually scheduled on grids.

How do we access the grid ?

- Command line with tools that you'll use
- Specialised applications
  - Ex: Write a program to process images that sends data to run on the grid as an inbuilt feature.
- Web portals
  - I2U2
  - SIDGrid

Grid Middleware glues the grid together

- *A short, intuitive definition:*

  the software that glues together different clusters into a grid, taking into consideration the socio-political side of things (such as common policies on who can use what, how much, and what for)

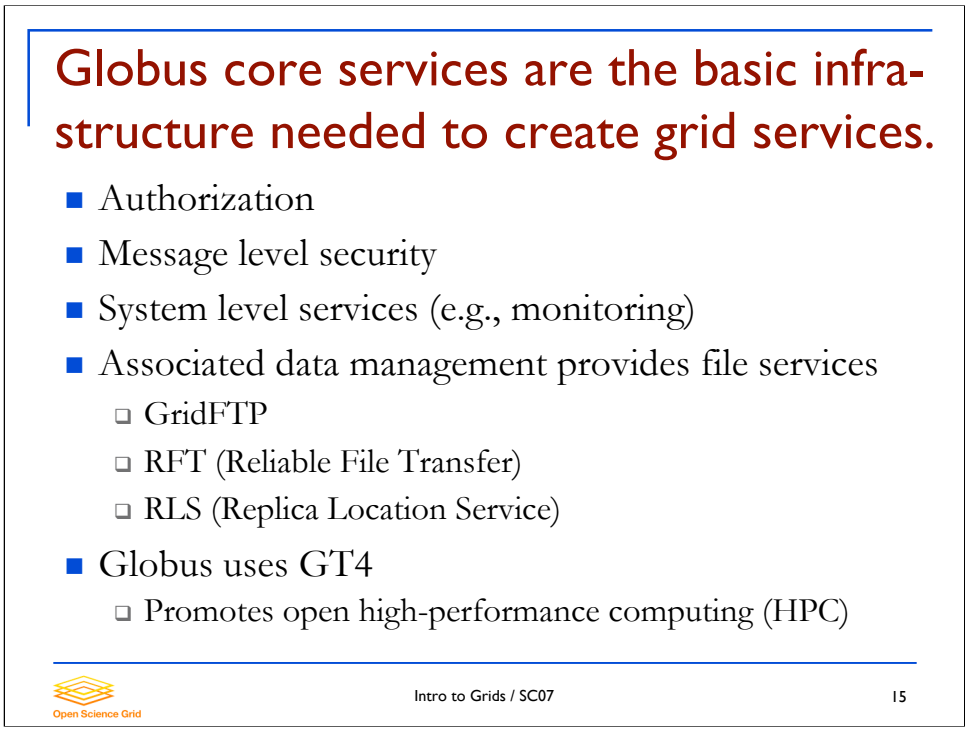
GM services couple users with remote resources through resource brokers.

- Offers services that couple users with remote resources through resource brokers
- Remote process management
- Co-allocation of resources
- Storage access
- Information
- Security
- QoS

Globus Toolkit is the *de facto* standard for grid middleware.

- Developed at ANL & Uchicago (Globus Alliance)
- Open source
- Adopted by different scientific communities and industries
- Conceived as an open set of architectures, services and software libraries that support grids and grid applications
- Provides services in major areas of distributed systems:
  - Core services
  - Data management
  - Security

Note: Our very own Ben has worked on and with Globus for more than 5 years.



GridFTP: a high performance, secure, and reliable data transfer protocol based on the standard FTP. GridFTP enhances FTP with GSI security, multiple data channels for parallel transfers, third party transfers, and authenticated reusable channels.

RFT: protocol that provides reliability and fault tolerance for file transfers.

RLS: a component of the data grid architecture. It provides access to mapping information from logical names to physical names of items. It's goal is to reduce access latency, improve data locality, improve robustness, scalability and performance for distributed applications

RLS produces replica catalogs (LRCs), which represent mappings between logical and physical files scattered across the storage system. For better performance, the LRC can be indexed..

Roadmap for this tutorial

- **Execution:** running programs with GRAM and Condor
  - GRAM is a Globus Toolkit component
- **Data management:** moves data with the grid
- **Information systems:** give users info about the grid to:
  - Decide where to run jobs
  - Find out job, network status, etc
- **Security:** authentication, authorization & accounting
- **National Grids:** Open Science Grid (OSG) and TeraGrid
- **Workflow**

Data movement:

- input data: move to execution node
- output data: move from execution node locally (laptop)
- GridFTP

Security is general (not related to a specific component, but integrated within the entire grid)

it isn't really a separate software component like the software talked about in the previous sections are. instead, its pervasive throughout all the software. There are three broad topics here: authentication (who are you), authorization (what are you allowed to do?) and accounting (what have you done?). All of this needs to be done in a decentralised grid setting, which can make it difficult.

The lab exercises will use both grids, OSG and TG.

Job and resource management

- Compute resources have a **local resource manager** (LRM) that controls:
  - Who is allowed to run jobs
  - How jobs run on a specific resource
- GRAM
  - Helps running a job on a remote resource
- Condor
  - Manages jobs

This component deals with job submission.

Naturally, users want to be able to runs their programs – for example, data processing, simulation. This section provides you with the answer on how this task can be accomplished on grids.

We can run our jobs on other computers on the grid to make use of those resources.

The challenging part is that sites on the grid have different platforms, therefore a uniform way of accessing the grid must be deployed. This is exactly the purpose of the grid software stack - it tries to deal with some of those differences.

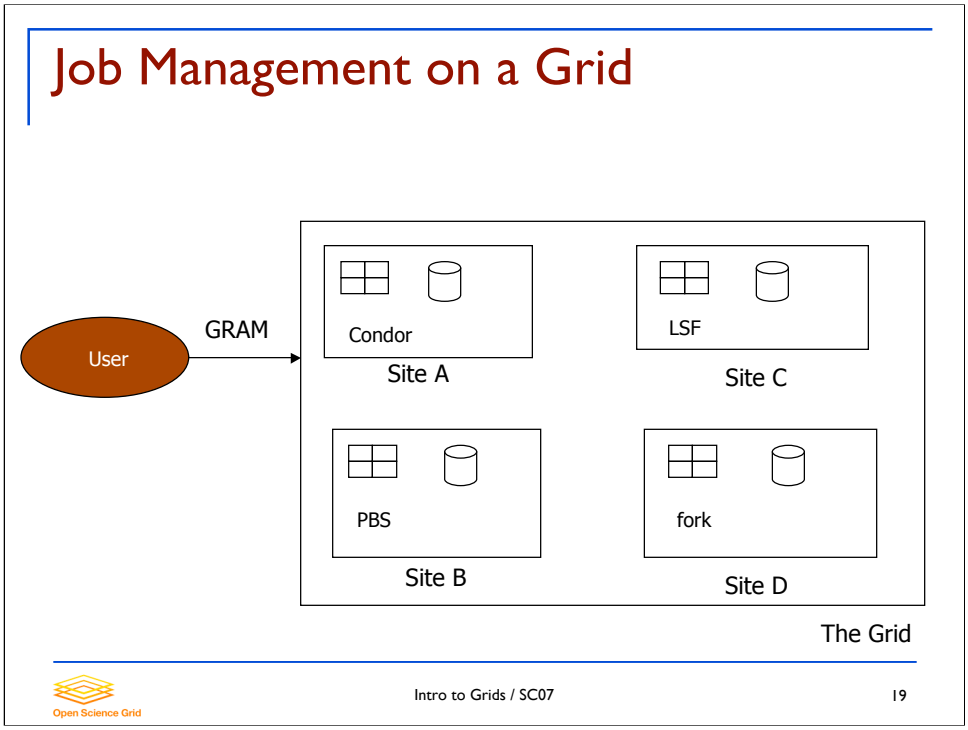
Local Resource Managers control which, when and where jobs run.

- LRM is software on a compute resource
- Controls on which jobs can run on which processors, and when
- *Example policy:*
  - Each cluster node can run one job.
  - If there are more jobs, then they must wait in a queue
- LRMs allow nodes in a cluster can be **reserved** for a specific person
- *Examples:* PBS, LSF, Condor

So on a particular site, we have a few components involved in job submission.

Closest to the hardware, we have a piece of software called a Local Resource Manager. This is a piece of software which manages the local resource, by specifying the order and location of jobs. In its simplest form, it makes sure only one user is using each compute node at once, and has some kind of queue.

More advanced LRMs can deal with reservations, prioritising between different users and various other parameters. LRMs are not grid specific – an LRM manages jobs on a particular cluster. It is important to realize that an LRM doesn't deal with jobs coming in from elsewhere. You need to, for example, log in directly to the cluster to submit your jobs.



Each cluster has a local resource manager on it. Each site can have a different local resource manager, chosen by the site administration. What we need to do now is to be able to access these resources in a homogeneous way -- this can be done my placing a new layer between the user (submission node) and each cluster. In the globus world, this is accomplished by piece of the globus toolkit called GRAM – the Grid Resource Allocation Manager.

GRAM provides a common network interface to various LRMs – so users can go to any site on your grid and make a job submission without really needing to know which LRM is being used. But GRAM doesn't manage the execution – its up the the LRM to do that. Make sure you understand the distinction between the resource allocation manager (GRAM) and the LRM (for example Condor).

GRAM and the LRM work together – GRAM gets the job to a cluster in a standard way for execution, and the LRM then executes it.

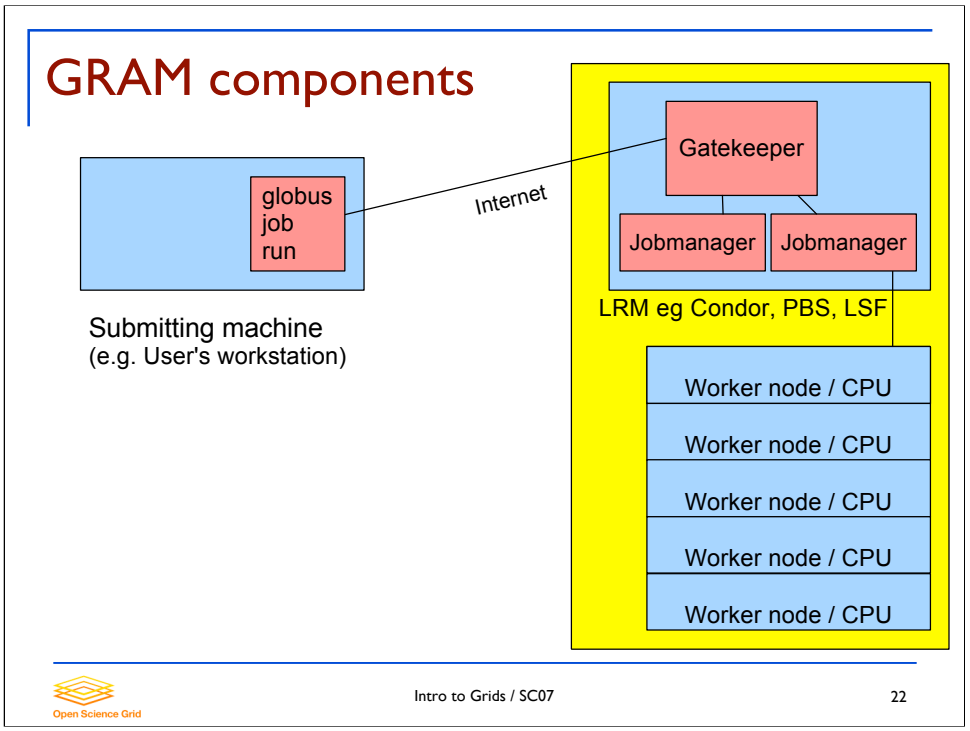
http://www.globus.org/toolkit/docs/4.0/execution/

GRAM provides a standardised interface to submit jobs to LRMs.

- **GRAM** = Globus Resource Allocation Manager
- Clients submit a job request to GRAM
- GRAM translates into something a(ny) LRM can understand
- Same job request can be used for many different kinds of LRM

GRAM's abilities

- Given a job specification:
  - Create an environment for a job
  - Stage files to and from the environment
  - Submit a job to a local resource manager
  - Monitor a job
  - Send notifications of the job state change
  - Stream a job's stdout/err during execution


GRAM components
globus
job
run
Internet
Gatekeeper
Jobmanager
Jobmanager
LRM eg Condor, PBS, LSF
Worker node / CPU
Worker node / CPU
Worker node / CPU
Worker node / CPU
Worker node / CPU
Submitting machine
(e.g. User's workstation)
Open Science Grid
Intro to Grids / SC07
22

Condor is a software system that creates an HTC environment.

- Created at UW-Madison
- Detects machine availability
- Harnesses available resources
- Uses remote system calls to send R/W operations over the network
- Requires no account login (?) on remote machines
- Provides powerful resource management by matching resource owners with consumers (broker)

Condor is a specialized workload management system for compute-intensive jobs.

Like other full-featured batch systems, Condor provides a job queueing mechanism, scheduling policy, priority scheme, resource monitoring, and resource management. Users submit their serial or parallel jobs to Condor, Condor places them into a queue, chooses when and where to run the jobs based upon a policy, carefully monitors their progress, and ultimately informs the user upon completion.

Condor - features

- Checkpoint & migration
  - Why is it important?
- Remote system calls
  - Able to transfer data files and executables across machines
- Job ordering
- Job requirements and preferences can be specified via powerful expressions

Condor lets you manage a large number of jobs.

- Specify the jobs in a file and submit them to Condor
- Condor runs them and keeps you notified on their progress
  - Mechanisms to help you manage huge numbers of jobs (1000's), all the data, etc.
  - Handles inter-job dependencies (DAGMan)
- Users can set Condor's job priorities
- Condor administrators can set user priorities
- Can do this as:
  - Local resource manager (LRM) on a compute resource
  - Grid client submitting to GRAM (as Condor-G)

Commonly when you submit a job to the grid, you do it as a large number of separate job submissions – for example if you want to use 100 CPUs – you might make a few hundred job submissions then, each one intended to use its own CPU. This can get a bit hard to manage, so you can use a piece of software called Condor – this can (amongst many other things) manage large numbers of jobs for you.

It can track their progress, restart them if they fail, handle dependencies between different jobs so that you can say that some jobs must not start till some other job has finished.

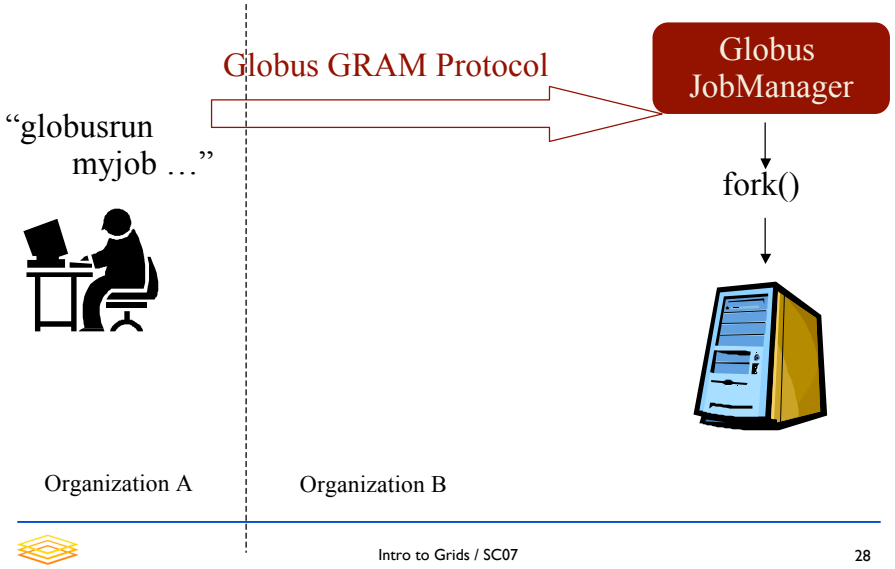
Condor-G is the job management part of Condor.

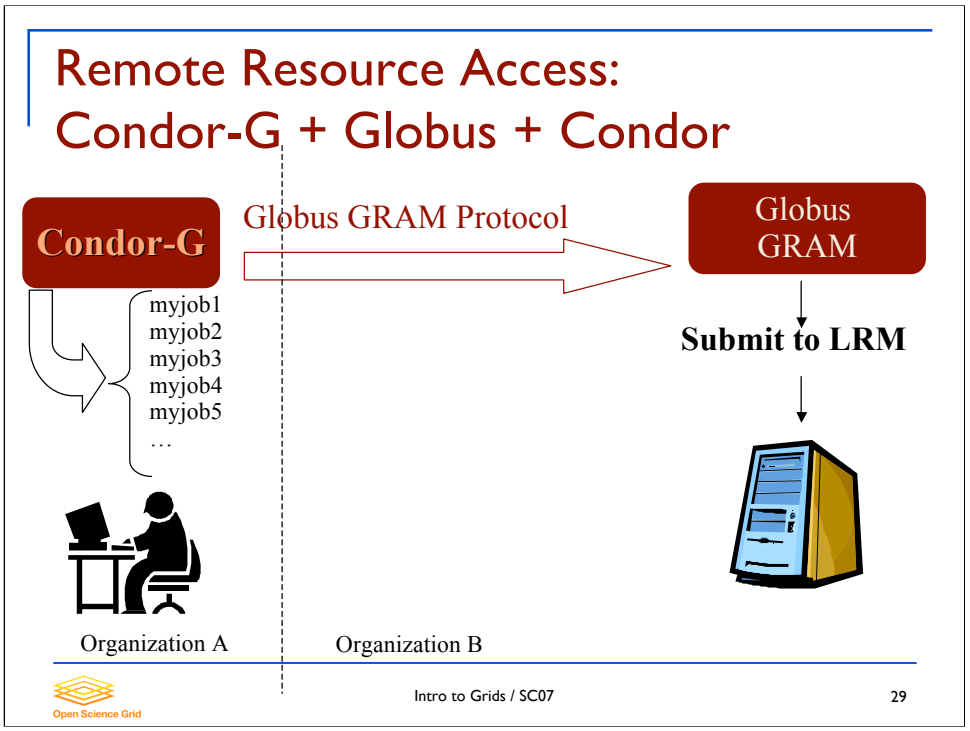
- *Hint:* Install Condor-G to submit to resources accessible through a Globus interface.
- Condor-G does not ***create*** a grid service.
- It only deals with ***using*** remote grid services.

Condor-G does whatever it takes to run your jobs, even if …

- The gatekeeper is temporarily unavailable
  - Gatekeeper =
- The job manager crashes
- Your local machine crashes
- The network goes down

Remote Resource Access: Globus




Remote Resource Access:
Condor-G + Globus + Condor
Condor-G
Globus GRAM Protocol
Globus GRAM
myjob1
myjob2
myjob3
myjob4
myjob5
…
Submit to LRM
Organization A
Organization B
Open Science Grid
Intro to Grids / SC07
29

Data Management

- Want to move data around:
  - Store it long term in appropriate places (e.g., tape silos)
  - Move input to where your job is running
  - Move output data from where your job ran to where you need it (eg. your workstation, long term storage)
- Exercises will introduce Globus Toolkit component called GridFTP

What we have covered so far relates to jobs and execution.

One of the other very big areas in using the grid is dealing with large amounts of data.

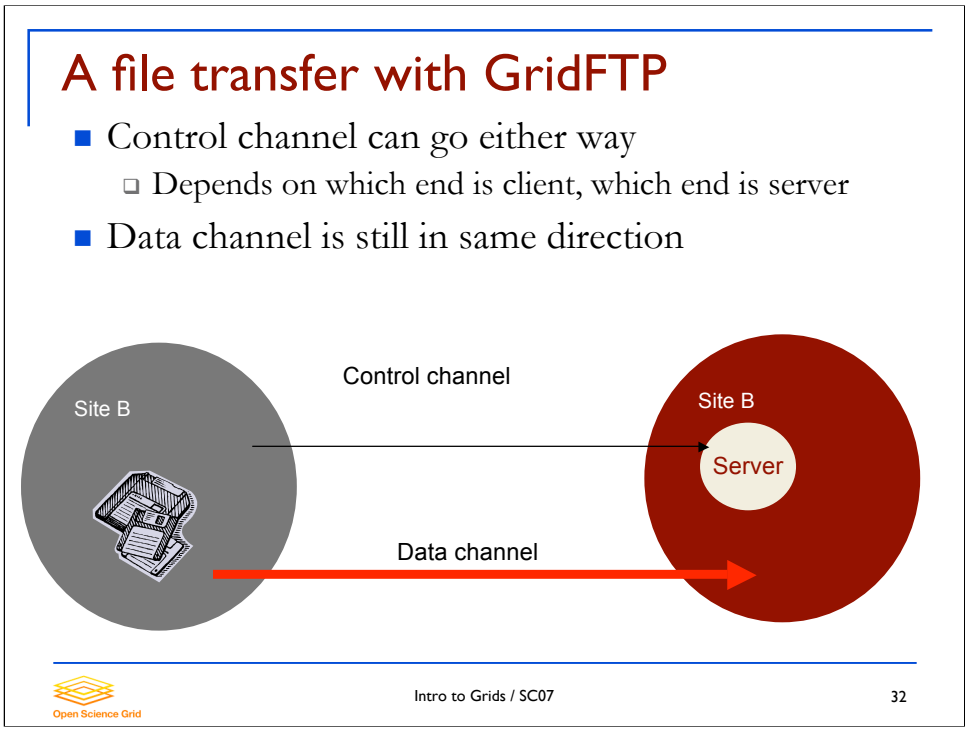
We need to deal with things such as moving data around, storing it somewhere long term, keeping track of where we've stored it, for large amounts (terabytes, and heading towards petabytes) and large numbers of files.

So, like the GRAM case, we have some grid software to do this for us. For data movement, pretty much the standard that everyone uses is GridFTP.

There are various other pieces of software to deal with storing data within a site (like an LRM but for data) such as dCache; and for keeping track of where you've put data (such as the Globus Replica Location Service).

High-performance tools needed to solve several data problems.

- The huge raw volume of data:
  - Storing it
  - Moving it
  - Measured in terabytes, petabytes, and ???
- The huge number of filenames:
  - 10^{12} filenames is expected soon
  - Collection of 10^{12} of anything is a lot to handle efficiently
- How to find the data



We have a file on site A, and we want to move it to site B.

Let's assume there is a GridFTP server on site B, and we are running at site A.

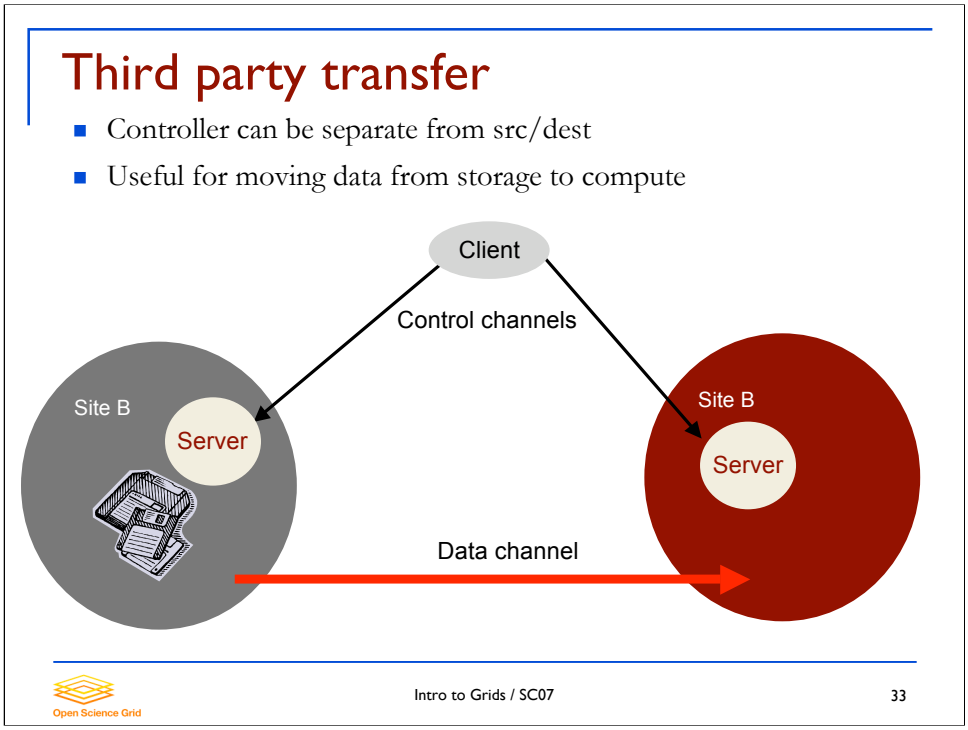
We can use a GridFTP client on site A (such as the globus-url-copy command) to copy from A to B.

What actually happens when we run a globus-url-copy command?

First the client establishes what is called a 'control channel'. This is a connection that will be used to co-ordinate data transfers.

Then we open a 'data channel' from the client to the server. The control channel is used to co-ordinate this between the client and the server.

And then, when the data channel is open, we send the contents of our data file through this channel, and the data arrives at the server, where it is written to disk. These operations constitute a single data transfer.

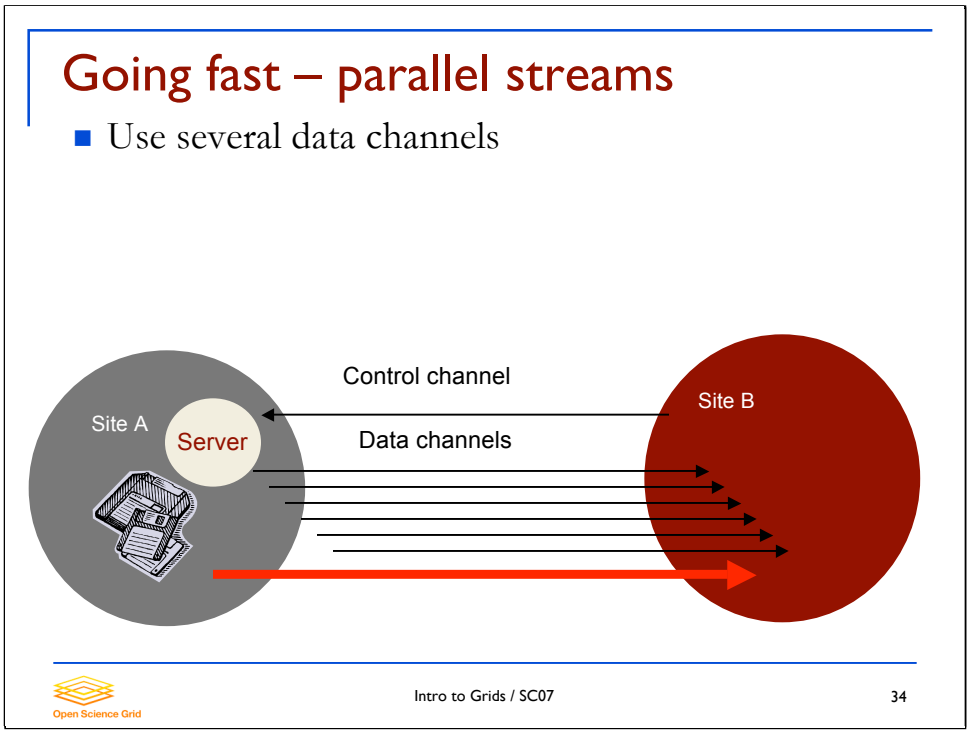


Third party transfer is a file is transferred from one server to another server without flowing through the client.

This is very useful when there is a very good connection between the two servers, and not such a good connection from those servers to you.

One way of transferring files would be to copy them from site A to your computer; and then from your computer to site B. But it would often be better to transfer directly from site A to site B.

Third party transfer does this. The scenario follows these steps: our client opens two control channels – one to each of the servers; then using those control channels it can get the two servers to establish a data channel between them and move the data across.



GridFTP has various other modes for increasing speeds.

For example, it can do 'parallel streams'. With parallel streams, we establish several data channels, and send the file in pieces across all of the connections. There are a few advantages here. The underlying network protocol, TCP, that most data transfers on the internet use, tries to share bandwidth equally between each stream. By using more streams, we can get more shares.

Some people think this is cheating, though and some people will get upset about it as a form of network abuse.

Another advantage is that if one or more of the channels do not run at full speed (for example, a single lost packet can cause massive slow down), this slowdown is ameliorated.

To make GridFTP go really fast:

- Use fast disks/filesystems
  - Filesystem should read/write > 30 MB/second
- Configure TCP for performance
  - See the TCP Tuning Guide at http://www-didc.lbl.gov/TCP-tuning/
- Patch your Linux kernel with web100 patch
  - Important work-around for Linux TCP "feature"
  - See http://www.web100.org
- Understand your network path

So here are some hints for experts from the experts.

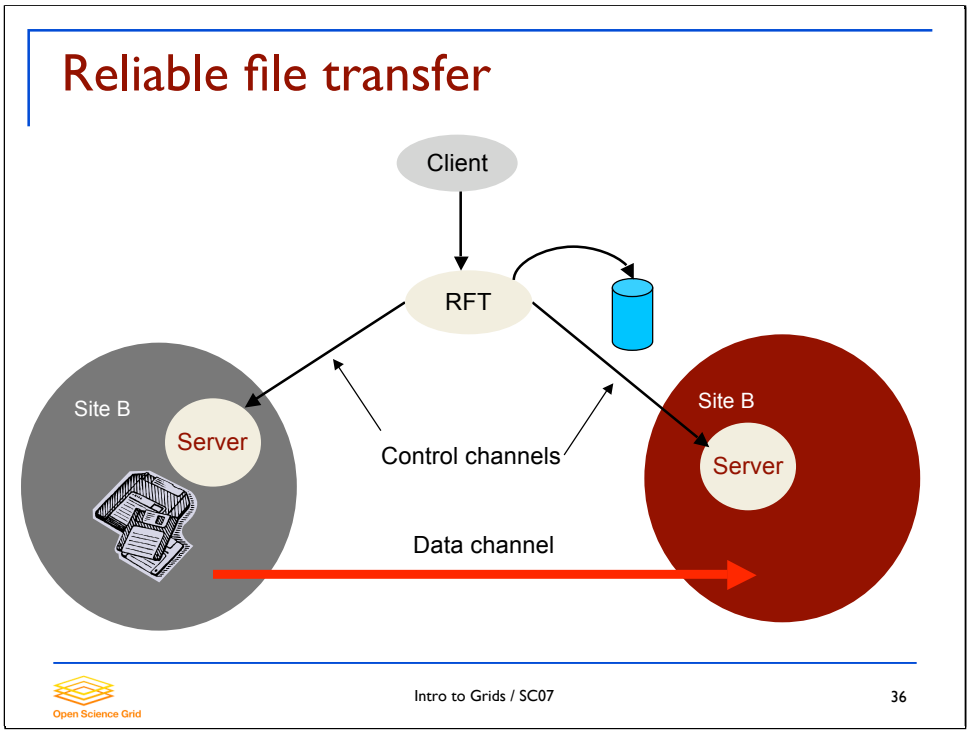
To make GridFTP go fast, you have to tune it.

You need to have some fast disks and filesystems at both ends of your connection.

You need to tune your TCP connections – TCP has many settings that can be diddled which affect performance. This is something of a black art.

There are some linux kernel patches called the web100 patches, which give more control and information about network connections on your servers.

And its important to understand what the network path between your two servers are – a big part of tuning your connections relies on knowing what the network looks like.



Just as in the job submission section, we mentioned a facility, GRAM, for submitting jobs , and then another program, Condor, sitting on top of that that provides things like reliability. A similar situation in the data transfer context.

We have a component called RFT, the Reliable File Transfer service, which comes as part of the Globus Toolkit.

http://www.globus.org/toolkit/docs/4.0/data/rft/

RFT acts as a client to GridFTP, providing management of a large number of transfer jobs – RFT can keep track of the state of each job, run several transfers at once, deal with connection failure, network failure, failure of any of the servers involved.

RFT

- WS-RF compliant High Performance data transfer service
  - Soft state
  - Notifications/Query
- Reliability on top of high performance provided by GridFTP
  - Fire and Forget
  - Integrated Automatic Failure Recovery
    - Network level failures
    - System level failures, etc.

Globus Replica Location Service maps logical filenames to physical filenames.

- Logical Filenames
  - Names a file with interesting data in it
  - Doesn't refer to location (which host, or where in a host)
- Physical Filenames
  - Refers to a file on some filesystem somewhere
  - Often use `gsiftp://` URLs to specify
- Two RLS catalogs: Local Replica Catalog and Replica Location Index

Local Replica Catalog (LRC) stores mappings from LFNs to PFNs.

- Interaction:
  - *Q:* Where can I get filename 'experiment_result_1'?
  - *A:* You can get it from gsiftp://gridlab1/home/benc/r.txt
- Undesirable to have one of these for whole grid
  - Lots of data
  - Single point of failure

Replica Location Index (RLI) stores mappings from LFNs to LRCs.

- Interaction:
  - *Q:* Who can tell me about filename 'experiment_result_1'.
  - *A:* You can get more info from the LRC at gridlab1
  - (Then go to ask that LRC for more info)
- Failure of one RLI or LRC doesn't break everything
- RLI stores reduced set of information, so can cope with many more mappings

Grid Information Systems

- Why do we want information?
  - Site selection
  - manual / automatic
- We can obtain such information via:
  - VORS in OSG
  - MDS in TG

The grid is a big place, in terms of whats going on – its hard to keep track of everything. Often you need to make decisions, such as which site to send your job to, or even which sites are available to send your job to. To do that, you need information; information about what your applications need, and information about what the grid can provide.

There are more specialised info systems like RLS.

Virtual Organizations (VO)

- Virtual Organization (classic definition)
  - Geographically distributed organization whose members are connected by common interests, and which communicate and coordinate their work through information services
  - Decentralized, non-hierarchical structures
- VO in the grid context
  - Facilitated by advancements by communication technologies
  - Grid computing enables distributed heterogeneous systems to work together as a single virtual system
  - OSG VO definition and list of existing VOs
    - In the lab, you will become a (temporary) member of the OSGEDU VO

Site Selection - Manually

- VORS = Virtual Organization Resource Selector

One of the things that is a fairly obvious thing to do, but that turns out to be quite hard in practise is what is know as 'site selection'.

This is quite a simple idea – you have jobs you want to run on the grid, there are various sites available and you want to choose which site or sites you are going to use. This can be done automatically or it can happen manually.

Manually, you might go to VORS and have a look and choose which sites look right based on your understanding of the sites and what your application needs.

There is a whole area of research into doing site selection automatically. It turns out to be very hard to do in general for a number of reasons.

Site Selection - Automatically

- Abstract job description
  - Site selection and data source selection done via programs
- Let the programs decide:
  - Where to run programs
  - Where to get data
- Swift and Pegasus have 'site selectors'
  - Pieces of code written in Java
  - Gives abstract description: "I want to run 'convert'"
  - Returns more concrete: "Run convert on site X"
- DAGman → Condor matchmaking

If your 'building block' task descriptions are sufficiently abstract that you could run on different grid sites, we now have the scope for our workflow system to choose which sites to run on. (we haven't covered workflows !)

Pegasus and Swift both have the concept of 'site selectors' – these are pieces of code that are written in Java that you give an 'abstract' job description to, and that gives back a more concrete decision about where that job should run.

DAGman uses Condor which has matchmaking.

Good site selection is hard.

- Some workflow systems to provide plug in points
- Actual useful site selectors are difficult to write – area of research
- Easy to come up with simple selectors:
  - Constant
  - Round robin
  - Random
- Difficult to write a site selector that does better

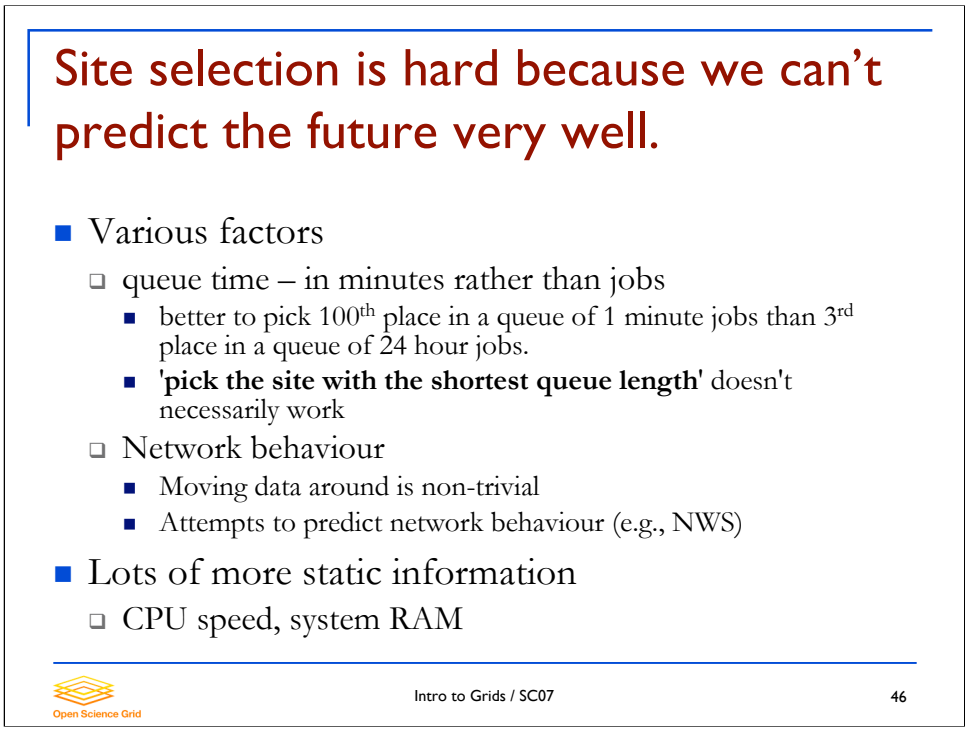
Workflow systems can do site selection. However, the actual process of deciding which site to run on or to take data from turns out to be very hard.

Workflow systems provide the places for you to specify site selection, but they don't provide decent selection algorithms.

Easy to come up with simple selectors – for example, constant selector that always chooses the same site; round robin that picks sites in its list entirely at random; the random selector that picks sites at random. These are simple and not very good.

Its difficult to write a better one, though.

Next slide discusses that.



Site selection is hard mostly because we can't predict the future very well.

Some of the factors that we can't easily predict:

The queue time for a job – we can see the queue length and so perhaps see that we are 10th in the queue. But that doesn't take into account the time that those jobs will run, or that that there may be higher priority queues. For example, it might be better to take 100th place in a queue with 100 x 1minute jobs compared to taking 3rd place in a queue with 3 x 24 hour jobs. So 'pick the shortest queue' doesn't work.

Network behaviour – grid workflows often involve moving around a lot of data – its difficult to know how the network between various points will behave. There are some tools such as the Network Weather Service (NWS) that try to help there.

Conversely, there's lots of static information that you can predict because its static – eg., CPU speed, RAM.

ReSS

- ReSS is a lightweight Resource Selection Service for push-based job handling systems
- implements cluster-level Workload Management on OSG.
- ReSS is deployed on OSG 0.6.0 and used by FermiGrid
- More info at http://osg.ivdgl.org/twiki/bin/view/ResourceSelection/
- Also see BDII and CeMON at http://is.grid.iu.edu/documentation.html

Motivations for ReSS

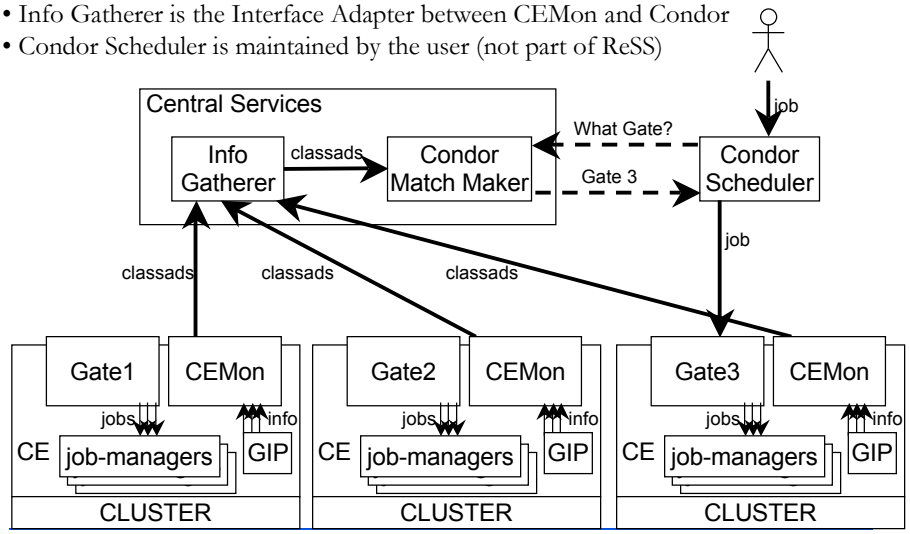
- Implement a light-weight cluster selector for push-based job handling services
- Enable users to express requirements on the resources in the job description
- Enable users to refer to *abstract* characteristics of the resources in the job description
- Provide soft-registration for clusters
- Use the standard characterizations of the resources via the Glue Schema

Technology

- ReSS basis its central services on the Condor Match-making service
  - Users of Condor-G naturally integrate their scheduler servers with ReSS
  - Condor information collector manages resource soft registration
- Resource characteristics is handled at sites by the gLite CE Monitor Service (CEMon)
  - CEmon registers with the central ReSS services at startup
  - Info is gathered by CEMon at sites running Generic Information Prividers (GIP)
  - GIP expresses resource information via the Glue Schema model
  - CEMon converts the information from GIP into old classad format. Other supported formats: XML, LDIF, new classad
  - CEMon publishes information using web services interfaces

Architecture

- Info Gatherer is the Interface Adapter between CEMon and Condor
- Condor Scheduler is maintained by the user (not part of ReSS)



Grid Security

- Identity and Authentication
- Message Protection
  - Confidentiality
  - Integrity
- Authorization
- Single Sign On
- Accounting

This section will talk about some of the various topics involved in grid security.

Identity and authentication – asking “who are you?”;

Message protection – that is making sure that either

other people can't read what you are sending over the network, or

other can't tamper with it;

Authorization – asking “what are you allowed to do?”; single sign on – that means you only have to type your password in once and you can use the whole grid without typing it again; and accounting - “what have you done?”

Message Protection

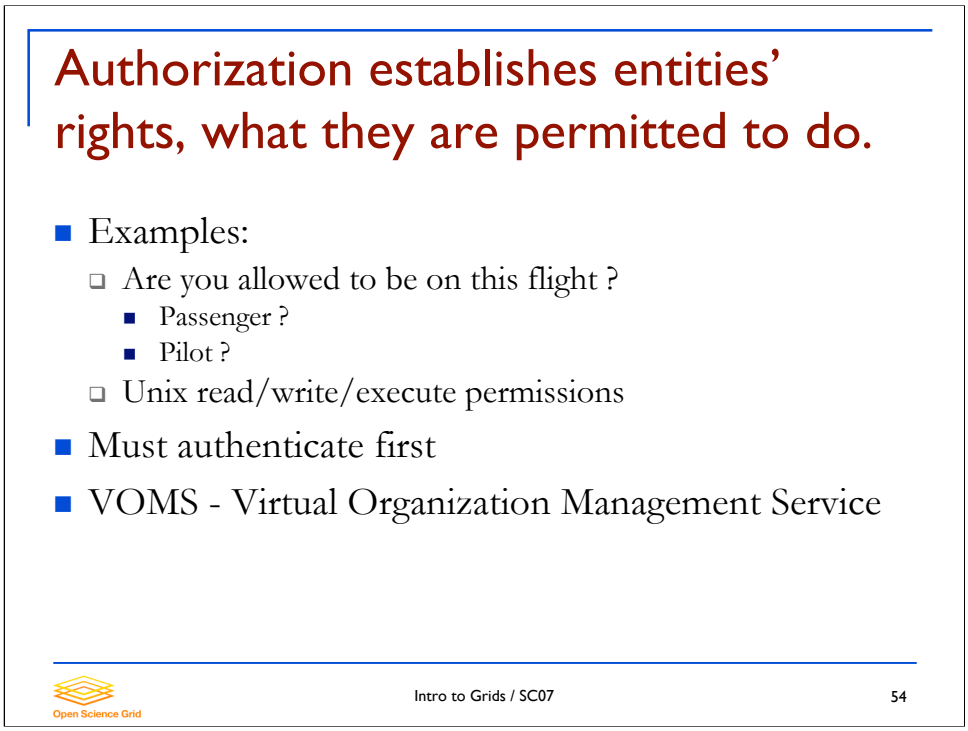
Authentication establishes an entity's identity.

- Each entity should have an **identity**
- Is the entity who he claims he is?
- *Examples:*
  - Driving License
  - Username/password
- Stops masquerading impostors

One of the fundamental concepts in grid security is "identity". This basically means asking the question "Who are you?".

We give every 'entity' on the grid an identity; we give it a unique name, distinct from any other entity's name; and we give it a way to provide that it really has that name. An entity is a person, or a computer or a service.

Each of these identities gets a name called an X.509 subject name; and it has a credential to prove that it owns that name. That's authentication – proving who you are.



Once you've authenticated yourself to some grid site, you need to be authorized to perform certain actions. What should ad should not you be allowed to do? *Example*: Should you be allowed to run jobs or transfer data or store files?

This is authorization.

A real life example is, having shown your ID card at an airport to prove your name, are you then allowed to get on any particular flight? That's what your e-ticket is for, not your passport; your passport binds you to your name, and your e-ticket says that the person with your name is allowed to board.

In the grid context, you authenticate to prove your subject name, and then authorization happens (or does not happen). Usually that means looking up in a map file to determine which unix user name you have access to.

In the exercises you'll see that on different machines, you will be mapped to different local unix users – some machines will give you your own account, some you will all share an account.

This is controlled ultimately by the owner of the resource – they say who is allowed to use their resource and who isn't.

[Talk about VOMS here and how it allows site owners to delegate mapping responsibility to others in a controlled manner.]

Single Sign-On (SSO) is a necessary function for complex Grid apps.

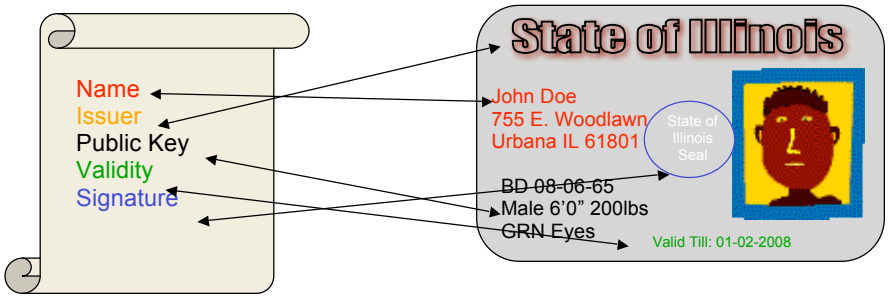
- Authenticate once rather than for every new access
- Enables easy coordination of varied resources
- Enables automation of process
- Allows remote processes and resources to act on user's behalf
- **Authentication** and **Delegation**

Single sign on (SSO) is a security concept where you log in once and have access to "everything" — rather than typing in a password every time you want to use a resource, you type it in once and that's good for some period of time (half a day, for example)

This is especially important in a grid setting because you're using so many different machines all over the place.

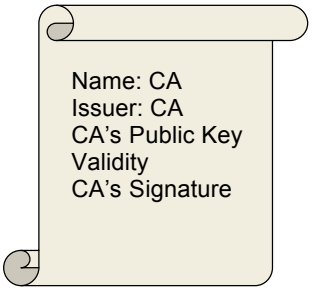
An X.509 certificate binds a public key to a name.

- Similar to passport or driver's license



A Certification Authority (CA) exists only to sign user certificates.

- The CA signs it's own certificate which is distributed in a trusted manner
- Verify CA certificate, then verify issued certificate



The CA private key is again a protected key.
But the CA public key is distributed widely.

Globus Security: The Grid Security Infrastructure (GSI)

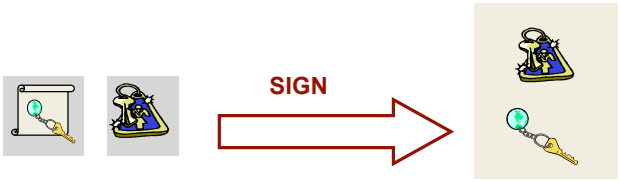
- A set of tools, libraries and protocols used in Globus to allow users and applications to securely access resources.
- Based on PKI
- Uses SSL for authentication and message protection
  - Encryption
  - Signature
- Adds Proxy Credentials and Delegation, needed for SSO

Each user has a set of GSI credentials to prove their identity on the grid

- Consists of a X.509 certificate and private key
- Long-term private key is kept encrypted with a passphrase
  - Good for security
  - Inconvenient for repeated usage

GSI Proxy Credentials provide the same effective ID as your certificate.

- Proxy credentials are short-lived credentials created by user
  - Proxy signed by certificate private key
- Short term binding of user's identity to alternate private key



Open Science Grid

Intro to Grids / SC07 60

Motivation:

- Encrypted private key requires some warm body,
- long term credential – prevent theft.

Same identity as the certificate proxy is created off

New key pair signed by your certificate

Clock synchronization issues

Proxy credentials are stored unencrypted for easy repeated access.

- Chain of trust
  - Trust CA → Trust User Certificate → Trust Proxy
- Key aspects
  - Generate proxies with short lifetime
  - Set the appropriate permissions on the proxy file
  - Destroy when done

Next: Delegation

GSI Delegation enables another entity to run as you.

- Provide the other entity with a proxy
- Ensure
  - Limited lifetime
  - Limited capability

Authorization components

- GUMS
- VOMS
- VOMRS

Useful to also consult the VO Alphabet Soup in the Documentation section https://twiki.grid.iu.edu/twiki/bin/view/Documentation/VOAlphabetSoup.

GUMS = Grid User Management System

- is a Grid Identity Mapping Service
- It maps the credential for each incoming job at a site to an appropriate site credential, and communicates the mapping to the gatekeeper.
- GUMS is particularly well suited to a heterogeneous environment with multiple gatekeepers;
- it allows the implemenation of a single site-wide usage policy, thereby providing better control and security for access to the site's grid resources. Read more at http://grid.racf.bnl.gov/GUMS/.

VOMS = Virtual Organization Membership Sevice

- is a system that manages real-time user authorization information for a VO
- designed to maintain only general information regarding the relationship of the user with his VO, e.g., groups he belongs to, certificate-related information, and capabilities he should present to resource providers for special processing needs.
- it maintains no personal identifying information besides the certificate. When a user submits a job, assuming the user is in good standing, VOMS also creates the necessary short-term credentials (extended proxy), required by grid resources before allowing the job to run.

A user who loses his or her authorization to use grid resources is deleted from the VOMS database and must be re-entered when the authorization is restored. VOMS, or an equivalent service, is required for OSG.

VOMRS = VO Management Registration Service

- major component of the extension to VOMS.
- VOMRS is a server that provides the means for registering members of a VO, and coordination of this process among the various VO and grid resource administrators
- maintains additional information on each VO member as required by individual grid resource providers, and some institution- and site-specific information.

It consists of a (separate) database to maintain user registration and institutional information, and a web user interface for input of data into the database and manipulation of that data. VOMRS populates the VOMS database via the VOMS administration package. VOMRS relies on the VOMS system to generate extended proxies for users as needed.Several OSG VOs are using VOMRS; it makes VO management much easier and more flexible.VOMRS is:・endorsed by the OSG management for use on OSG・part of the OSG software stack・used by the large VOs on OSG. VOMRS architecture diagram:

Accounting provides statistics regarding jobs that run on a grid.

- OSG accounting
  - Gratia

TeraGrid has more of this than OSG traditionally.

Grid Resources in the US

The OSG

Open Science Grid

Origins:

- National Grid (iVDGL, GriPhyN, PPDG) and LHC Software & Computing Projects

Current Compute Resources:

- 61 Open Science Grid sites
- Connected via Inet2, NLR.... from 10 Gbps – 622 Mbps
- Compute & Storage Elements
- All are Linux clusters
- Most are shared
  - Campus grids
  - Local non-grid users
- More than 10,000 CPUs
  - A lot of opportunistic usage
  - Total computing capacity difficult to estimate
  - Same with Storage

The TeraGrid

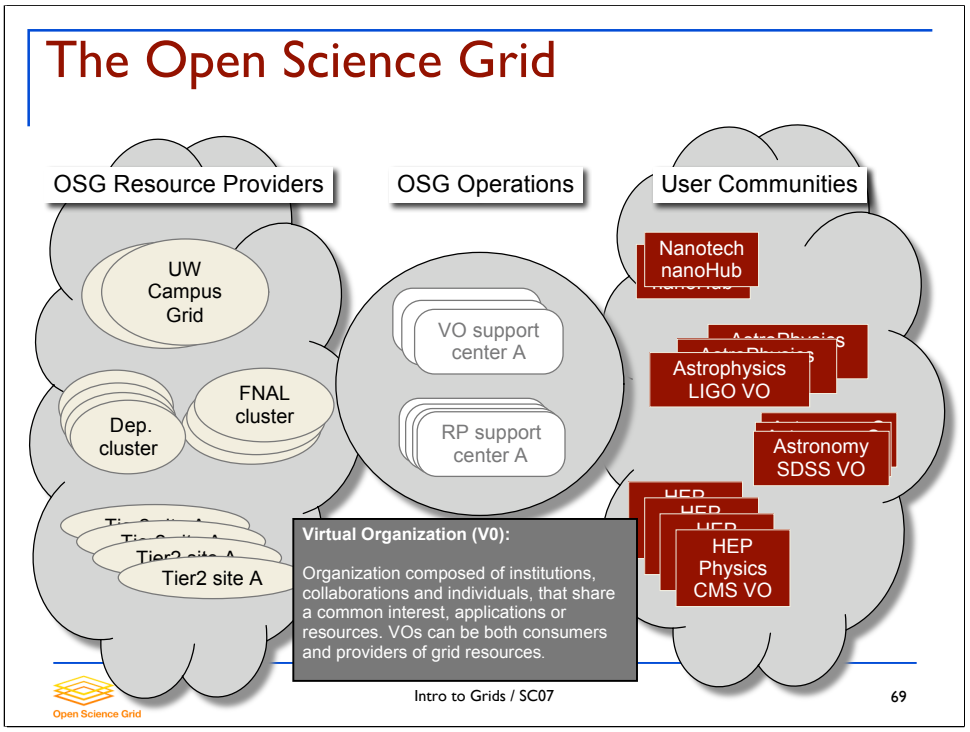
TeraGrid

Origins:

- National Super Computing Centers, funded by the National Science Foundation

Current Compute Resources:

- 9 TeraGrid sites
- Connected via dedicated multi-Gbps links
- Mix of Architectures
  - ia64, ia32: LINUX
  - Cray XT3
  - Alpha: True 64
  - SGI SMPs
- Resources are dedicated but
  - Grid users share with local and grid users
  - 1000s of CPUs, > 40 TeraFlops
- 100s of TeraBytes


The Open Science Grid
OSG Resource Providers
OSG Operations
User Communities
UW Campus Grid
FNAL cluster
Dep. cluster
Tier2 site A
VO support center A
RP support center A
Nanotech nanoHub
Astrophysics LIGO VO
Astronomy SDSS VO
HEP Physics CMS VO
Virtual Organization (V0):
Organization composed of institutions, collaborations and individuals, that share a common interest, applications or resources. VOs can be both consumers and providers of grid resources.
Open Science Grid
Intro to Grids / SC07
69

Workflow Systems

- Motivation
  - Grid tools
  - Job submission
  - Data transfer
- But an application requires more …

The ultimate goal of using the grid isn't to submit jobs and move data. Those are a means to an end.

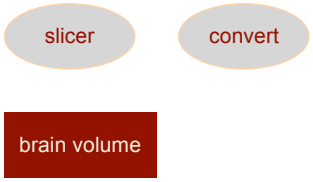
The bigger picture is that we would like to be able to build applications that use the grid (by running those tasks your self), instead of manually sending individual jobs to the grid and waiting for results.

Workflow ties pieces of an application together in standard ways.

- Better than doing it yourself
- Workflow systems handle many of the gritty details
  - You could implement them yourself
  - You would do it very badly
    - Trust me – even better, ask Miron
- Useful 'additional' functionality beyond basic plumbing such as providing **provenance**

A very simple example

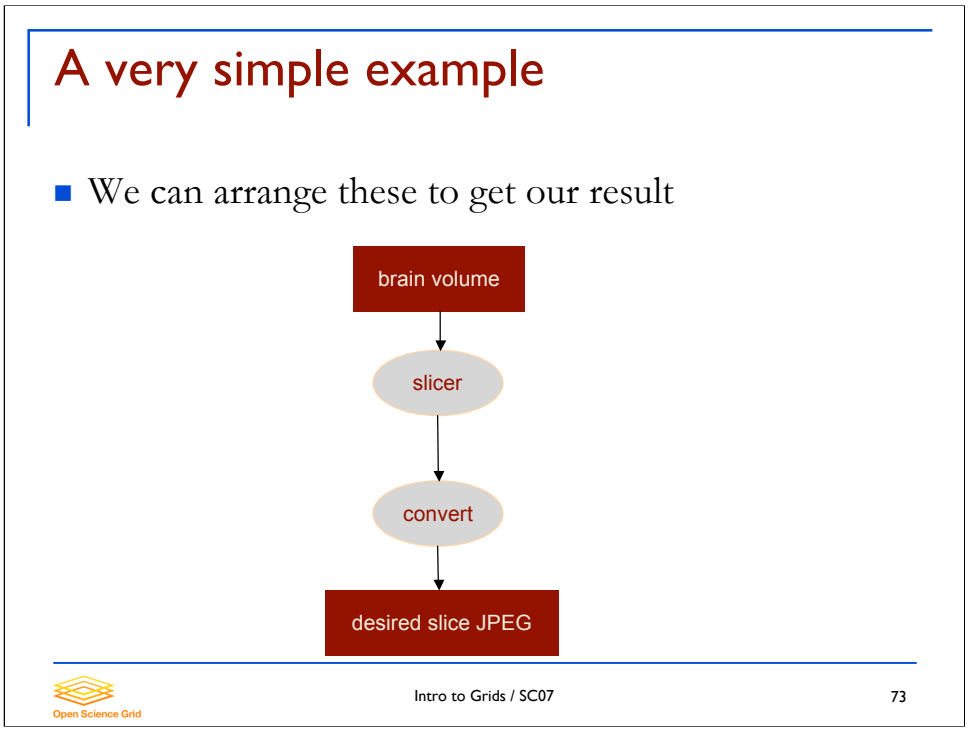
- What we have:
  - Two applications
  - Some data



- *Goal:* Produce a JPEG of a slice through the supplied brain.

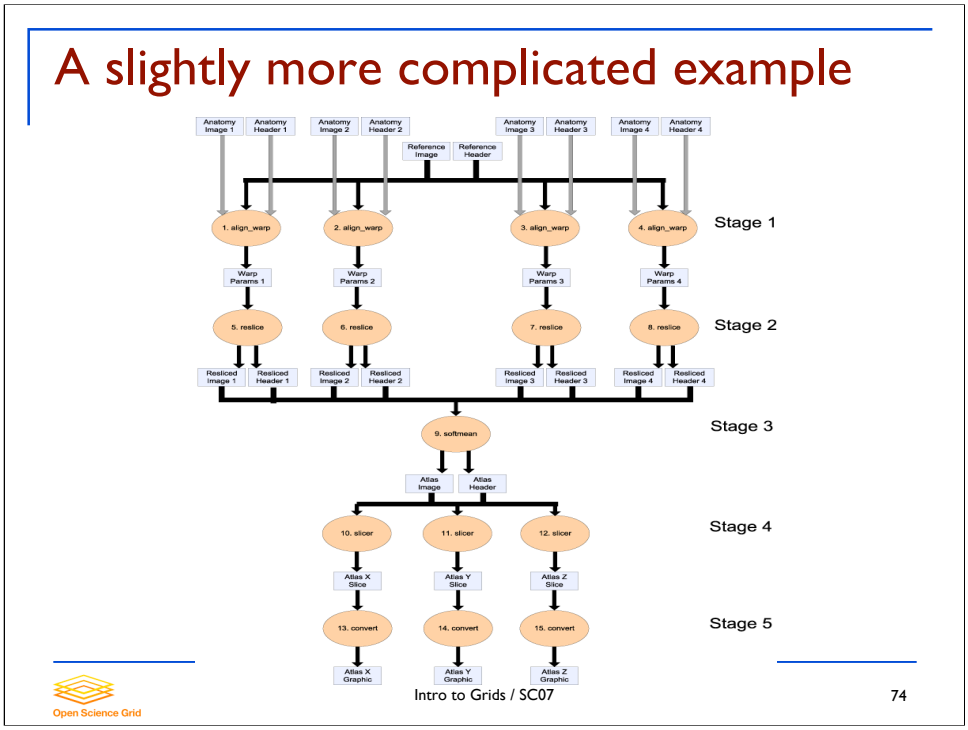
We have a data file, a 3D brain scan volume, and two programs – one of them, slicer, makes a 2D slice of a brain scan volume and outputs a .pnm file; the other can convert a PNM file into a JPEG. Conveniently, this is called 'convert'.

Perhaps we have a goal of producing a .jpeg of a brain slice. It should be fairly obvious what you need to do.

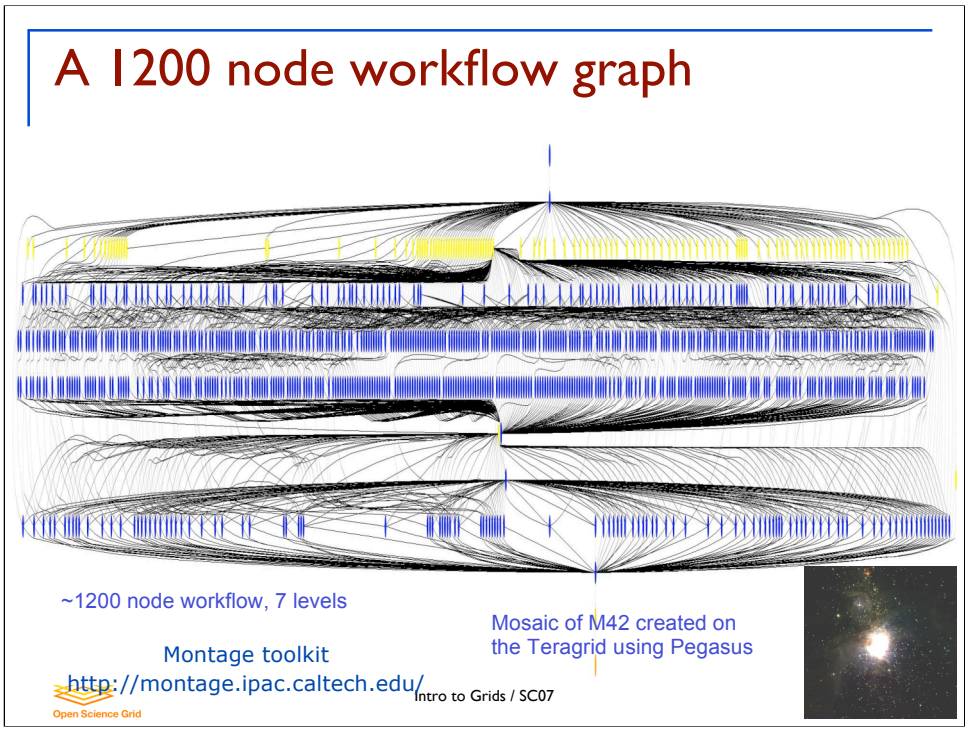


Here we have a (graphical representation of) what we need to do – start with data file, run slicer, run convert, end up with our data file. A workflow system would allow us to easily express this action plan.

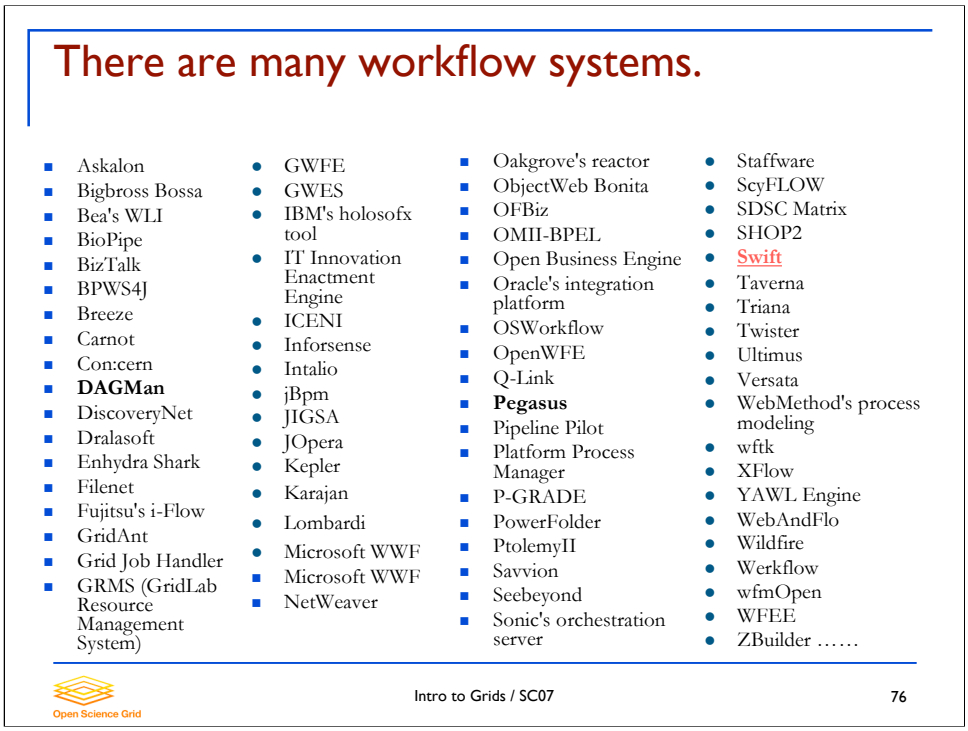
This is a very simple example – real grid workflows are much much more complicated and can handle complex scenarios; their usefulness is then fully justified.



Here we present a slightly more complicated example of of workflow: functional MRI brain processing. We start with a 3D brain volume and align, average, slice and generate 2D graphics based on the initial 3D model. Follow the arrows to get the plan of actions.



An even more complicated example: many of nodes, each represents a grid job, and many lines, representing dependencies.



There are a plethora other workflow systems, all different – if you thought there was a lot of middleware like globus, glite &c, workflow is even more so.

None of the workflow systems has established itself as the clear leader. We could point out DAGMan, Pegassus and Swift.

Workflows can be represented as graphs or programs.

- As graphs
  - DAGman
    - Visual representation
    - Straightforward visual representation for small workflows
  - Visual representation (flowcharts)
- As programs: Workflow language
  - Programming language specialised for 'scripting the grid'
  - Easy to bring in programming language concepts
    - variables, loops, subroutines

First we'll look at workflows-as-graphs. You've already experienced this twice.

Our simple example was a graph laid out on a 2D plane – this projection screen. The kind of representation shows the tasks to be performed and how they are related.

Secondly, DAGman workflows are represented as a more theoretical form of graph – you specify jobs as the nodes on a graph and dependencies between them as arcs. recall the G is DAGman stands for 'graph'. DAG == Directed Acyclic Graph (a cycle-free graph that specifies dependencies via arrows)

Another way of thinking about workflows is that a workflow is a program.

A workflow system is then a system for "scripting the grid" - so just like other programming languages are specialised for other purposes, like C for systems work, matlab for numerical work, visual basic for desktop office applications, bash shell for running scripts of unix executables.

Then we fairly naturally end up with programming languages constructs – variables, for loops, subroutines.

Swift is a dataflow language that specifies workflow and transformations.

- **Workflows** are specified in terms of data and **transformations** to be made to that data
  - Transform input files to output files using application code (unix executable)
- Facilitates site selection
- Easy to re-run failed jobs (in different place?)

A third example is Swift. Swift's 'building block' is data transformation

Swift knows what the input and output files are for each kind of transformation, because you define that as part of the workflow. You also need to tell swift where executables are for your transformations on different sites. Once you've done that, Swift has enough information to run transformations on different grid sites – it knows enough to stage in the data, run the job and stage the results back out; and it knows that because this is a dataflow language that it can try the job again somewhere else.

One of the useful things that you can do with this (that both Swift and Pegasus do) is site selection

Easy to rerun jobs – on same site or different site, because we have captured the effects of the job

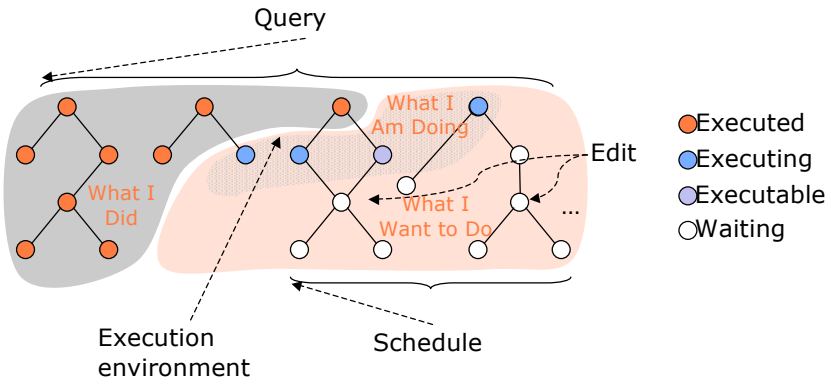
Pegasus has similar behaviour (due to shared ancestry with VDS)

Provenance tells you where results come from and how they were computed.

- Definition …
- Know what has been computed already
- Various ways to use this information
  - *Example:* In the graph pruning example earlier, we knew that some data had already been computed.

Provenance is a term from fine art.

Workflow specifies what to do; Provenance tracks what was done.

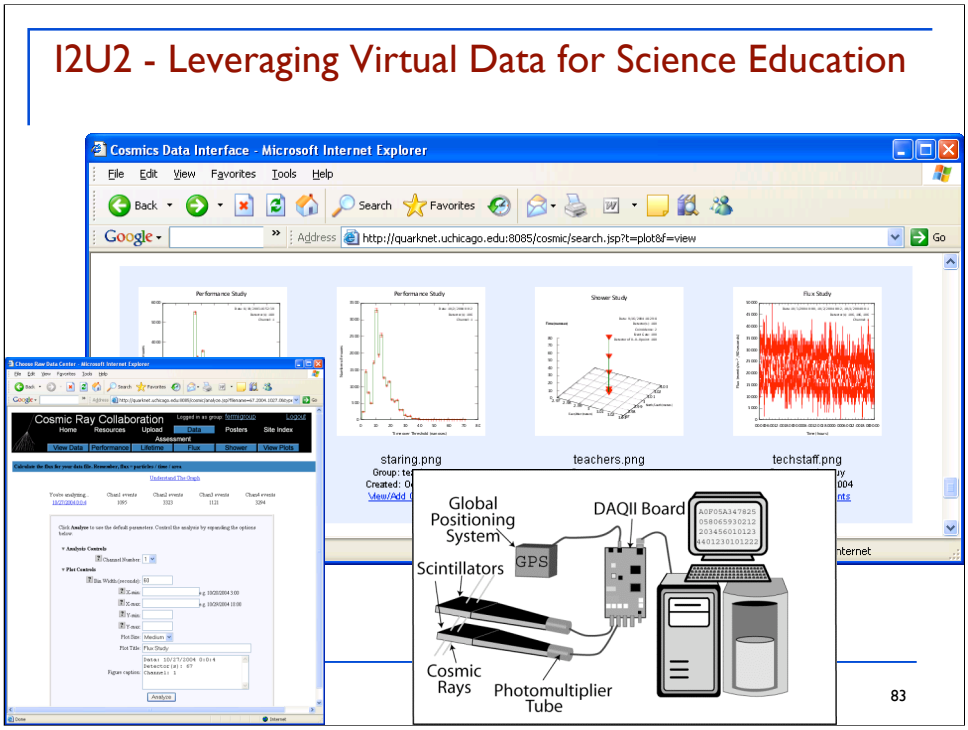


Things we can do with Provenance:

- Run the workflow again (maybe on different machines) and see if we get same results
- Find out how someone else computed a result
- Catalogue which results have been computed already
  - Optimise new workflows that are related
  - If intermediate results are used already, then we don't need to compute again.
- TODO notes: http://twiki.ipaw.info/bin/view/Challenge/FirstProvenanceChallenge

Nine Provenance Challenge Queries

- Find the process that led to Atlas X Graphic / everything that caused Atlas X Graphic to be as it is. This should tell us the new brain images from which the averaged atlas was generated, the warping performed etc.
- Find the process that led to Atlas X Graphic, excluding everything prior to the averaging of images with softmean.
- Find the Stage 3, 4 and 5 details of the process that led to Atlas X Graphic.
- Find all invocations of procedure align_warp using a twelfth order nonlinear 1365 parameter model (see model menu describing possible values of parameter "-m 12" of align_warp) that ran on a Monday.
- Find all Atlas Graphic images outputted from workflows where at least one of the input Anatomy Headers had an entry global maximum=4095. The contents of a header file can be extracted as text using the scanheader AIR utility.



More information about I2U2 …

What's next ?

- How to join OSG
  - Contact us: eot@opensciencegrid.org
    - Mailing lists
    - OSGEDU VO
      - Use OSG resources
      - Contribute OSG resources
  - Learn more
    - opensciencegrid.org/Education
  - Attend our Grid Schools
    - www.opensciencegrid.org/workshops
  - Host a grid school
  - Ideas for cooperation ?
    - Research, K12 grid education, …

Summary

What is a Grid?

- A Grid is a system that
  - coordinates resources that are not subject to centralized control
  - using standard, open, general-purpose protocols and interfaces
  - to deliver nontrivial qualities of service
- What is the difference between a job scheduler and a job manager ? Give examples of each.
  - A job scheduler is a system for submitting, controlling and monitoring the workload of batch jobs in one ore more computer. The jobs are scheduled fore execution at a time decided by the system according to an available policy and on availability of resources. Ex: Condor-G
  - A job manager's function is to provide a single interface for requesting and using remote system resources for the execution of jobs. Ex: GRAM ("remote shell with features")

Now from a different perspective (we have learned so much about grids now), we can answer this question …

Discussion session questions

What is the difference between a job scheduler and a job manager?

- *Give examples of each.*
- A **job scheduler** is a system for submitting, controlling and monitoring the workload of batch jobs in one ore more computer. The jobs are scheduled for execution at a time decided by the system according to an available policy and on availability of resources. *Ex:* Condor-G
- A **job manager**'s function is to provide a single interface for requesting and using remote system resources for the execution of jobs. *Ex:* GRAM ("remote shell with features")

Summarize the interaction between job schedulers and other grid middleware.

What are the components of grid middleware?

What is the difference between Condor and Condor-G?

- See:

http://www.cs.wisc.edu/condor/condorg/versusG.html

Big source of confusion.

HPC vs HTC

- *HPC* = High Performance Computing
- Tremendous amount of computing power over a short period of time
- Supercomputers - expensive, centralized

- *HTC* = High Throughput Computing
- Large amounts of computing power over a long period of time
- Use many, smaller, cheaper PCs

How is data management component implemented in Globus?

How do we choose the right scheduler?

Why do we talk about VOs in grid computing? Why do we need VOs?

- Grid computing enables and simplifies collaboration among members of a VO.

- Find the list of all OSG VOs
- Find the sites that the OSGEDU VO are contributing to the OSG grid.

Why are information systems important in the grid context?

How does the grid determine if you will can submit a certain job to a certain site?

- Explain in detail.

Where to get more information

- The notes for this talk have URLs throughout.
- This course is based on Open Science Grid's grid schools programme.
  - http://www.opensciencegrid.org/workshop for latest
- Email us:
  - Ben Clifford: benc@ci.uchicago.edu
  - Alina Bejan: abejan@ci.uchicago.edu
  - Mike Wilde: wilde@mcs.anl.gov