

Introduction to Makeflow and Work Queue

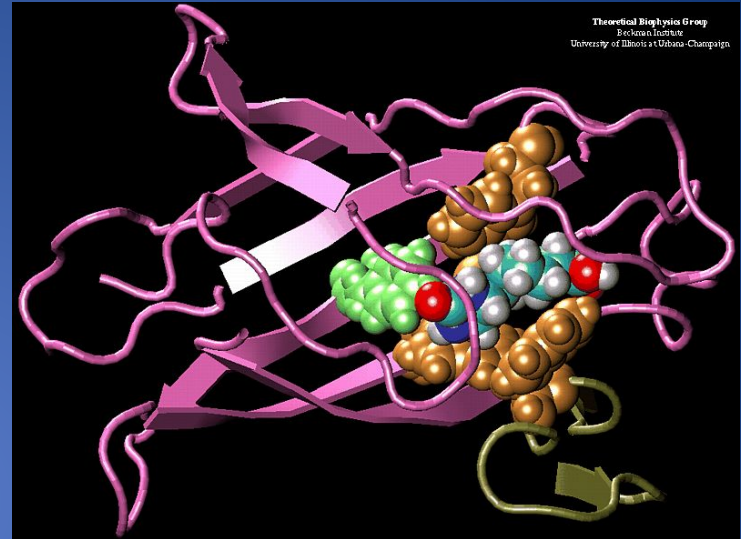
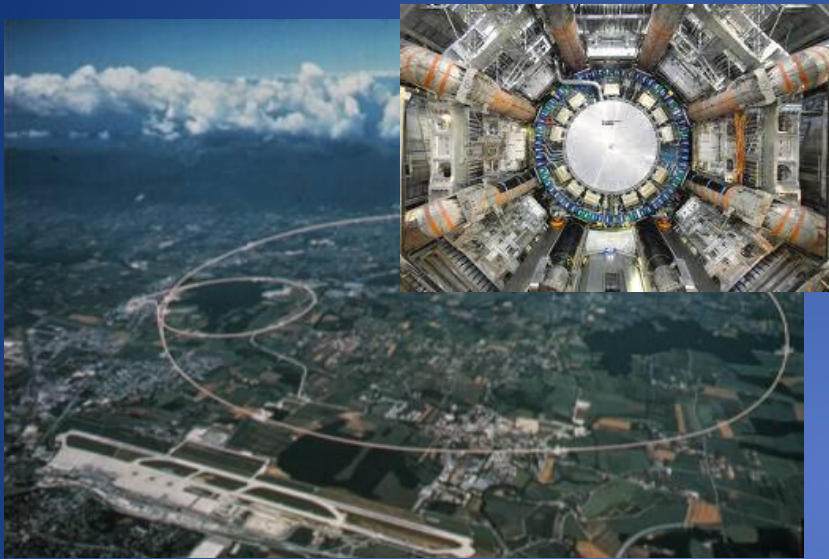
CSE 40822 – Cloud Computing – Spring 2016
Douglas Thain / Ben Tovar

The Cooperative Computing Lab

- We *collaborate with people* who have large scale computing problems in science, engineering, and other fields.
- We *operate computer systems* on the O (10,000) cores: clusters, clouds, grids.
- We *conduct computer science* research in the context of real people and problems.
- We *develop open source software* for large scale distributed computing.

<http://www.nd.edu/~ccl>

Science Depends on Computing!



AGTCCGTACGATGCTATTAGCGAGCGTGA

A screenshot of the Biometrics Research Grid (BYGRID) website interface. The browser window shows the URL "https://bygrid.cs.uiowa.edu/". The page title is "BYGRID - BIOMETRICS RESEARCH GRID". The interface includes a navigation menu on the left, a search and filter section at the top, and a main content area displaying a grid of face scan results. The search results show "Showing 1 to 10 of 93 results. Download all results as TXT or CSV or XML or TAB." The results are organized into columns for "Unvalidated", "Valid 1", "Valid 2", "Valid 3", and "Valid 4". Each result includes a small image of a face and a set of controls (Validate, Problem, Unvalidate) and a link to "record / preview / view / view all".

The Good News:
Computing is Plentiful!



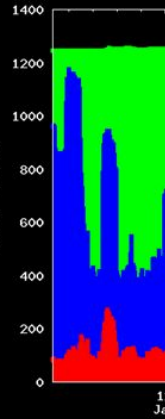
Eucalyptus
Systems



amazon
web services™

CPU Utilization for the Last Month

System Type	S	C	V	D
A: AIX				
D: DIX				
M: OS/2				
S: SUNOS				
H: HP/UX				
I: IRIX				
L: Linux				
W: Windows				



XSEDE

Extreme Science and Engineering
Discovery Environment

365688

466637

68086 (7%) CPU-Hours Used by Owner

900411 (100%) CPU-Hours Total



Windows Azure

NIMBUS

Big clusters are cheap!

The screenshot shows a web browser window with the URL <http://arstechnica.com/business/news/2011/09/>. The article title is "\$1,279-per-hour, 30,000-core cluster built on Amazon EC2 cloud" by Jon Brodtkin, published a day ago. Below the article is a CycleServer monitoring dashboard for Chef and Ganglia. The dashboard shows a table of nodes and a bar chart of completed converges.

\$1,279-per-hour, 30,000-core cluster built on Amazon EC2 cloud

By Jon Brodtkin | Published a day ago

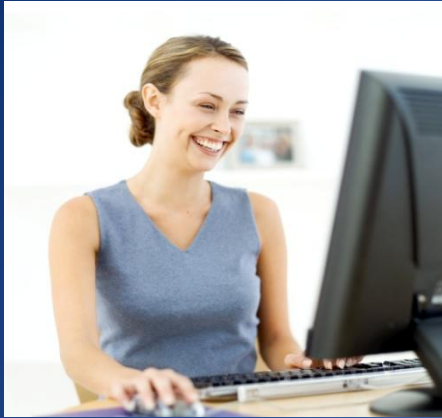
CycleServer Chef Ganglia

Show: **All** converges over the last **hour**

Host Name	Instance	Cluster	Status	Total Converges	Last Completed Convergence	Longest Convergence
ip-10-36-126-161.ec2.internal	i-b33abcd2	412		2	2011-07-30 15:27:42	3:50.787
ip-10-36-125-99.ec2.internal	i-591d9b38	412		4	2011-07-30 15:36:43	3:31.503
ip-10-36-125-91.ec2.internal	i-e522a484	412		4	2011-07-30 15:34:07	2:58.296
ip-10-36-125-137.ec2.internal	i-ff088e9e	412		3	2011-07-30 15:24:56	4:54.988
ip-10-35-9-95.ec2.internal	i-5d36b03c	412		2	2011-07-30 15:31:47	4:28.892
ip-10-35-3-63.ec2.internal	i-d70d8bb6	412		4	2011-07-30 15:28:33	4:10.597
ip-10-35-2-10.ec2.internal	i-e13c8a80	412		2	2011-07-30 15:21:27	3:36.483
ip-10-35-14-209.ec2.internal	i-a51492c4	412		3	2011-07-30 15:20:09	3:47.043
ip-10-35-10-207.ec2.internal	i-37399f56	412		2	2011-07-30 15:27:13	4:23.522

Completed Converges

hour
3:29 PM



I have a standard, debugged, trusted application that runs on my laptop.

A toy problem completes in one hour.
A real problem will take a month (I think.)

Can I get a single result faster?
Can I get more results in the same time?



Last year,
I heard about
this grid thing.

This year,
I heard about
this cloud thing.



What do I do next?



What they want.



What they get.



I can get as many machines
on the cloud as I want!

How do I organize my application
to run on those machines?

Our Philosophy:

- Harness all the resources that are available: desktops, clusters, clouds, and grids.
- Make it easy to scale up from one desktop to national scale infrastructure.
- Provide familiar interfaces that make it easy to connect existing apps together.
- Allow portability across operating systems, storage systems, middleware...
- Make simple things easy, and complex things possible.
- ***No special privileges required.***

A Quick Tour of the CCTools

- Open source, GNU General Public License.
- Compiles in 1-2 minutes, installs in \$HOME.
- Runs on Linux, Solaris, MacOS, Cygwin, FreeBSD, ...
- Interoperates with many distributed computing systems.
 - Condor, SGE, Torque, Globus, iRODS, Hadoop...
- Components:
 - Makeflow – A portable workflow manager.
 - Work Queue – A lightweight distributed execution system.
 - All-Pairs / Wavefront / SAND – Specialized execution engines.
 - Parrot – A personal user-level virtual file system.
 - Chirp – A user-level distributed filesystem.

<http://ccl.cse.nd.edu/software>

Install in Your Home Directory

```
cd $HOME
```

```
wget http://ccl.cse.nd.edu/software/files/  
cctools-4.2.2-source.tar.gz
```

```
tar xvzf cctools-4.2.2-source.tar.gz
```

```
cd cctools-4.2.2-source
```

```
./configure
```

```
make
```

```
make install
```

The image shows a stack of three overlapping browser windows. The top window displays the homepage of The Cooperative Computing Laboratory (CCL) at ccl.cse.nd.edu. The middle window shows the Makeflow software page, and the bottom window shows the Makeflow User's Manual page. The manual page is titled 'makeflow(1)' and contains the following sections:

- NAME**
makeflow - workflow engine for executing distributed workflows
- SYNOPSIS**
makeflow [options] <dagfile>
- DESCRIPTION**

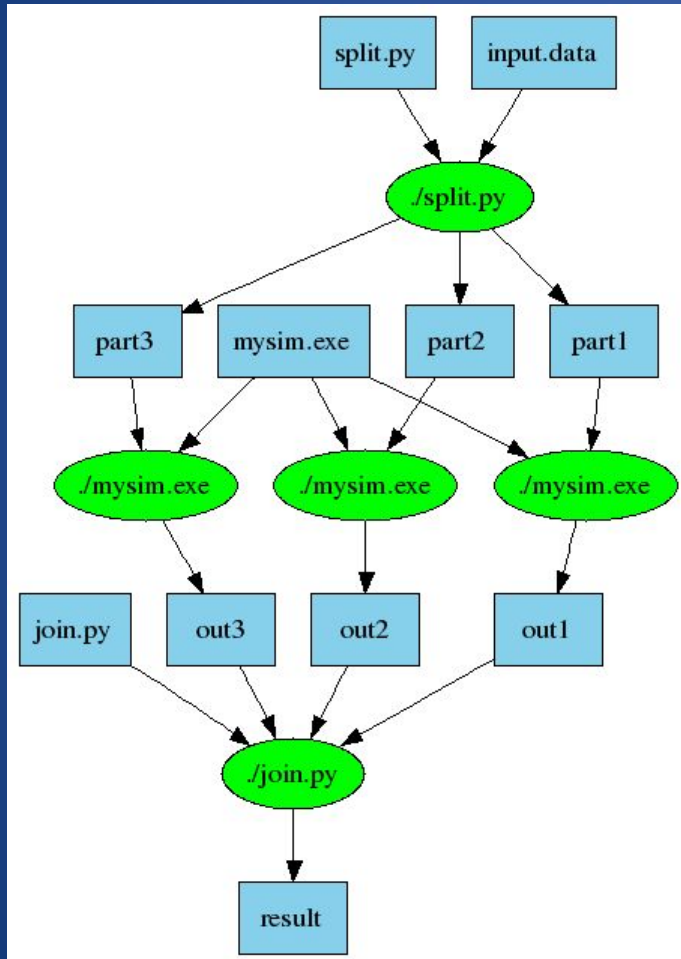
Makeflow is a workflow engine for distributed computing. It accepts a specification of a large amount of work to be performed, and runs it on remote machines in parallel where possible. In addition, **Makeflow** is fault-tolerant, so you can use it to coordinate very large tasks that may run for days or weeks in the face of failures. **Makeflow** is designed to be similar to Make, so if you can write a Makefile, then you can write a **Makeflow**.

You can run a **Makeflow** on your local machine to test it out. If you have a multi-core machine, then you can run multiple tasks simultaneously. If you have a Condor pool or a Sun Grid Engine batch system, then you can send your jobs there to run. If you don't already have a batch system, **Makeflow** comes with a system called Work Queue that will let you distribute the load across any collection of machines, large or small.
- OPTIONS**
When makeflow is ran without arguments, it will attempt to execute the workflow specified by the **Makeflow** dagfile using the local execution engine.
- Commands**
-c, --clean Clean up: remove logfile and all targets.
-f, --summary-log <file>
Write summary of workflow to file.

<http://ccl.cse.nd.edu>

Makeflow: A Portable Workflow System

An Old Idea: Makefiles



part1 part2 part3: input.data split.py

```
./split.py input.data
```

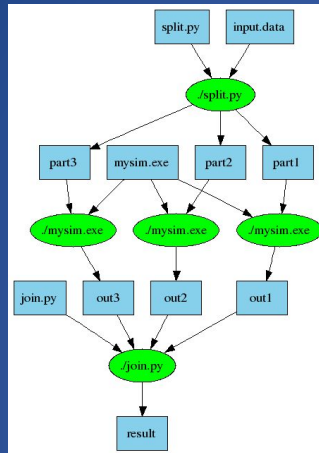
out1: part1 mysim.exe
./mysim.exe part1 >out1

out2: part2 mysim.exe
./mysim.exe part2 >out2

out3: part3 mysim.exe
./mysim.exe part3 >out3

result: out1 out2 out3 join.py
./join.py out1 out2 out3 > result

Makeflow = Make + Workflow



- Provides portability across batch systems.
- Enable parallelism (but not too much!)
- Trickle out work to batch system.
- Fault tolerance at multiple scales.
- Data and resource management.

Makeflow

Local

Condor

Torque

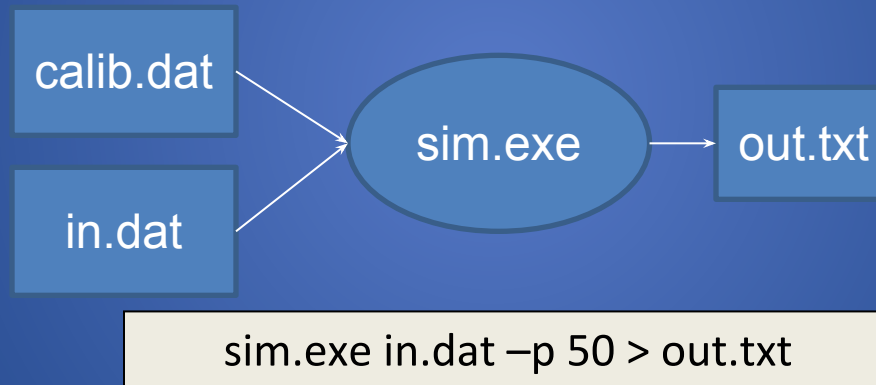
Work
Queue

<http://ccl.cse.nd.edu/software/makeflow>

Makeflow Syntax


[output files] : [input files]
[command to run]

One Rule



out.txt : in.dat **calib.dat sim.exe**

sim.exe -p 50 in.data > out.txt



You must state
all the files
needed by the command.

sims.mf

out.10 : in.dat calib.dat sim.exe

sim.exe -p 10 in.data > out.10

out.20 : in.dat calib.dat sim.exe

sim.exe -p 20 in.data > out.20

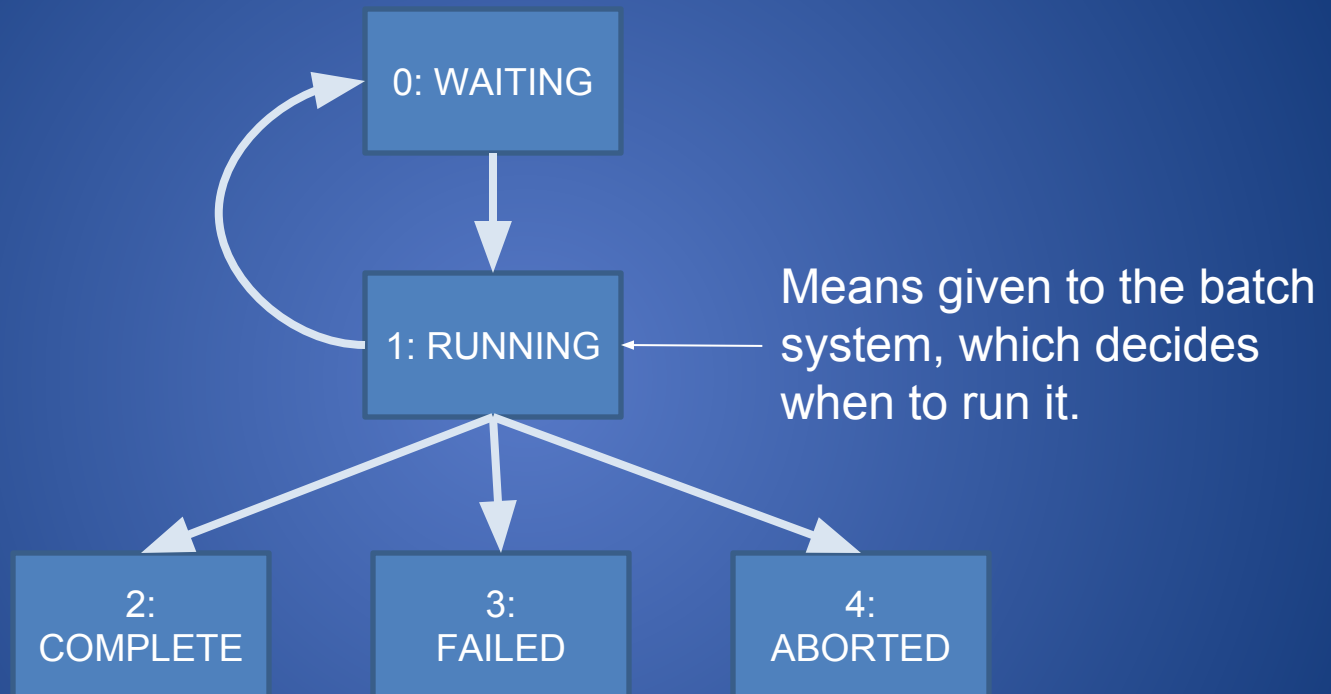
out.30 : in.dat calib.dat sim.exe

sim.exe -p 30 in.data > out.30

How to run a Makeflow

- Run a workflow locally (multicore?)
 - `makeflow -T local sims.mf`
- Clean up the workflow outputs:
 - `makeflow -c sims.mf`
- Run the workflow on Torque:
 - `makeflow -T torque sims.mf`
- Run the workflow on Condor:
 - `makeflow -T condor sims.mf`

Job States



Transaction Log

TIME, TASKID, STATE, JOBID, STATE[0], STATE[1], ...

1347281321284638 5 1 9206 5 1 0 0 0 6

1347281321348488 5 2 9206 5 0 1 0 0 6

1347281321348760 4 1 9207 4 1 1 0 0 6

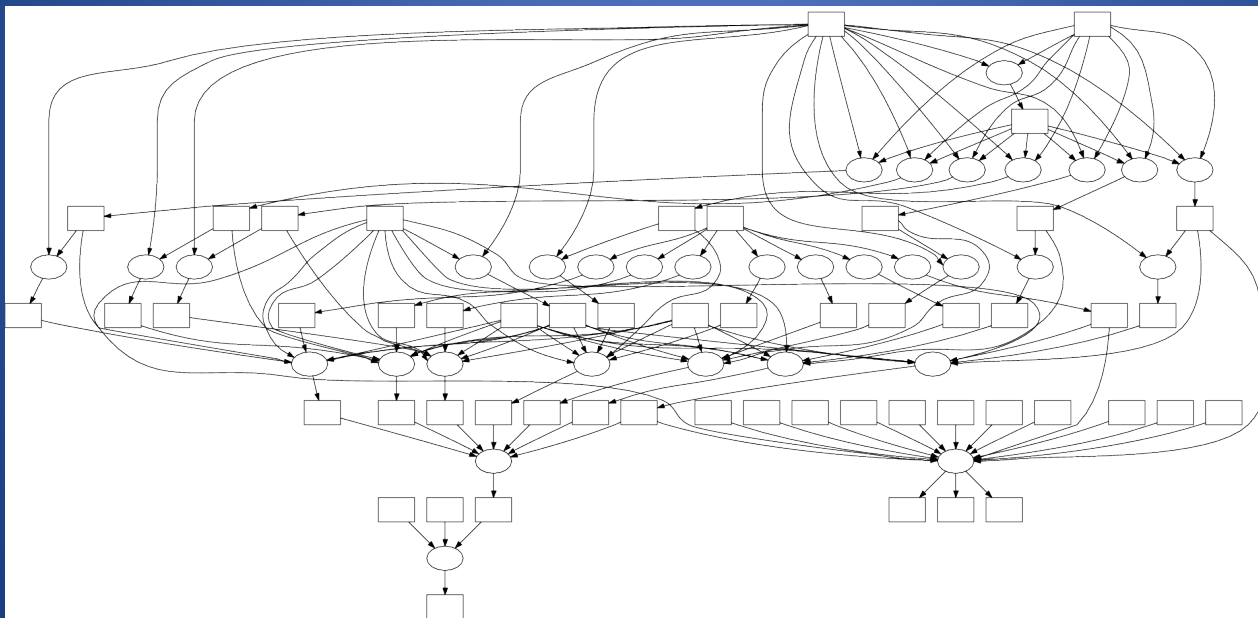
1347281321348958 3 1 9208 3 2 1 0 0 6

1347281321629802 4 2 9207 3 1 2 0 0 6

1347281321630005 2 1 9211 2 2 2 0 0 6

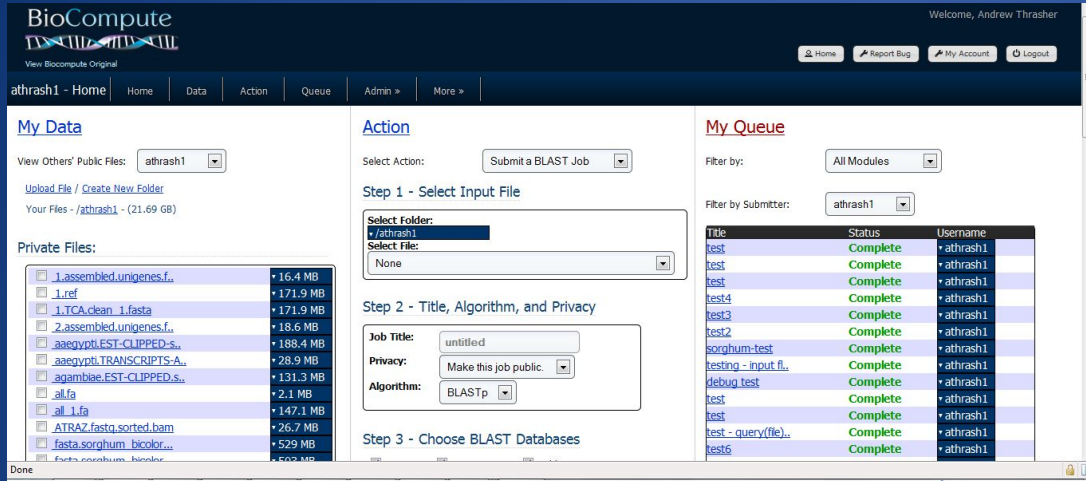
Visualization with DOT

- `makeflow_viz -D example.mf > example.dot`
- `dot -T gif < example.dot > example.gif`



DOT and related tools:
<http://www.graphviz.org>

Example: Biocompute Portal



The screenshot shows the Biocompute Portal interface. The top navigation bar includes 'Home', 'Data', 'Action', 'Queue', 'Admin', and 'More'. The 'My Data' section shows a list of files under 'Private Files' with columns for file name and size. The 'Action' section shows a dropdown menu set to 'Submit a BLAST Job' and a 'Select File' dropdown. The 'My Queue' section shows a table of jobs:

Title	Status	Username
test1	Complete	athrash1
test2	Complete	athrash1
test3	Complete	athrash1
test4	Complete	athrash1
test5	Complete	athrash1
test6	Complete	athrash1

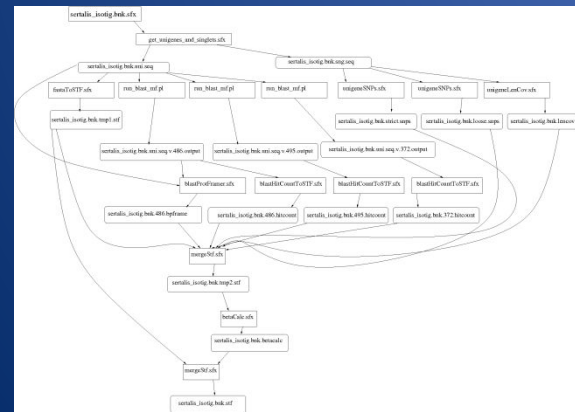


BLAST
SSAHA
SHRIMP
EST
MAKER
...

Generate Makefile

Progress Bar

Transaction Log



Run Workflow

Update Status

Make flow

Condor Pool

Submit Tasks

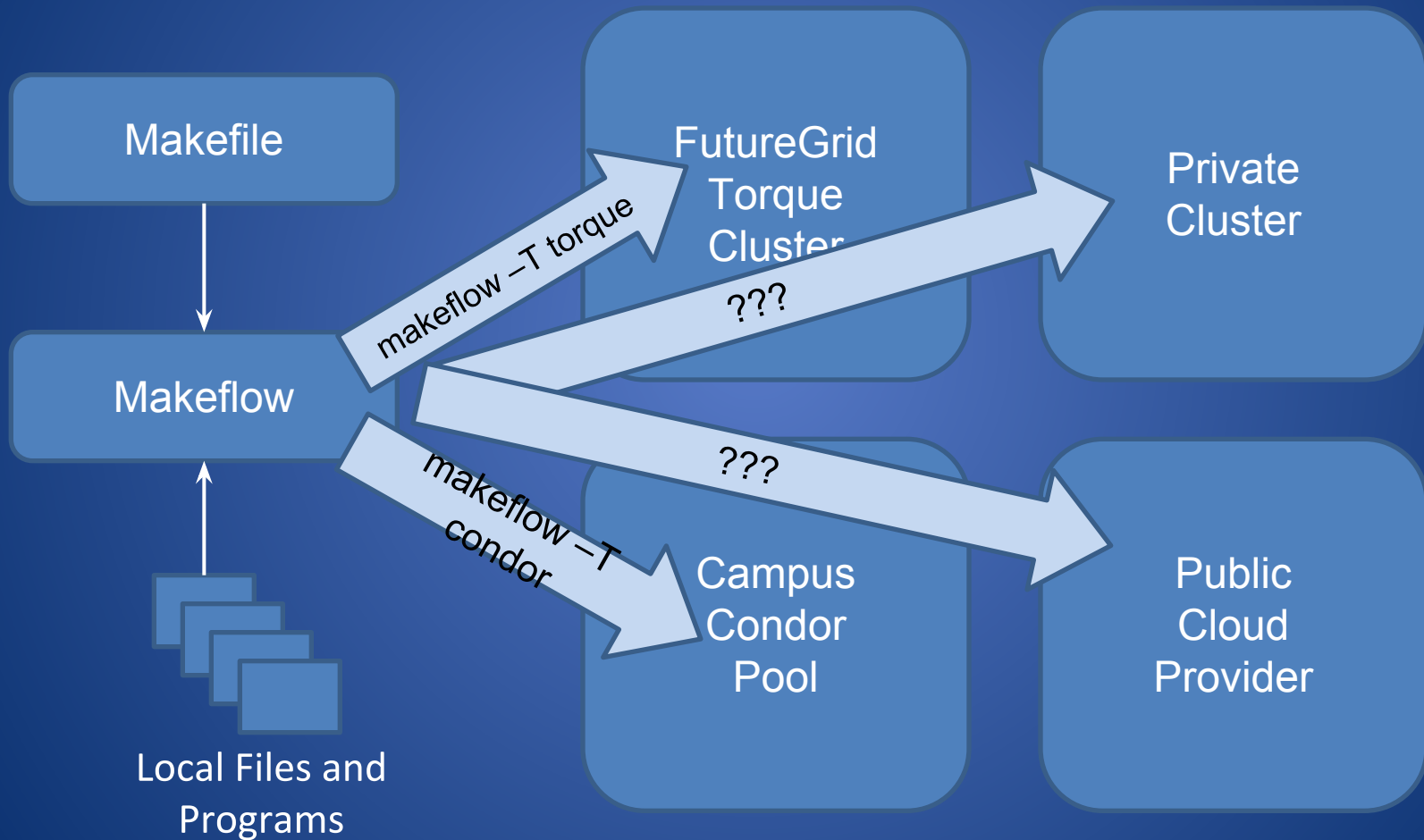
Makeflow Applications

The image illustrates a bioinformatics pipeline using Makeflow. It is divided into three main sections:

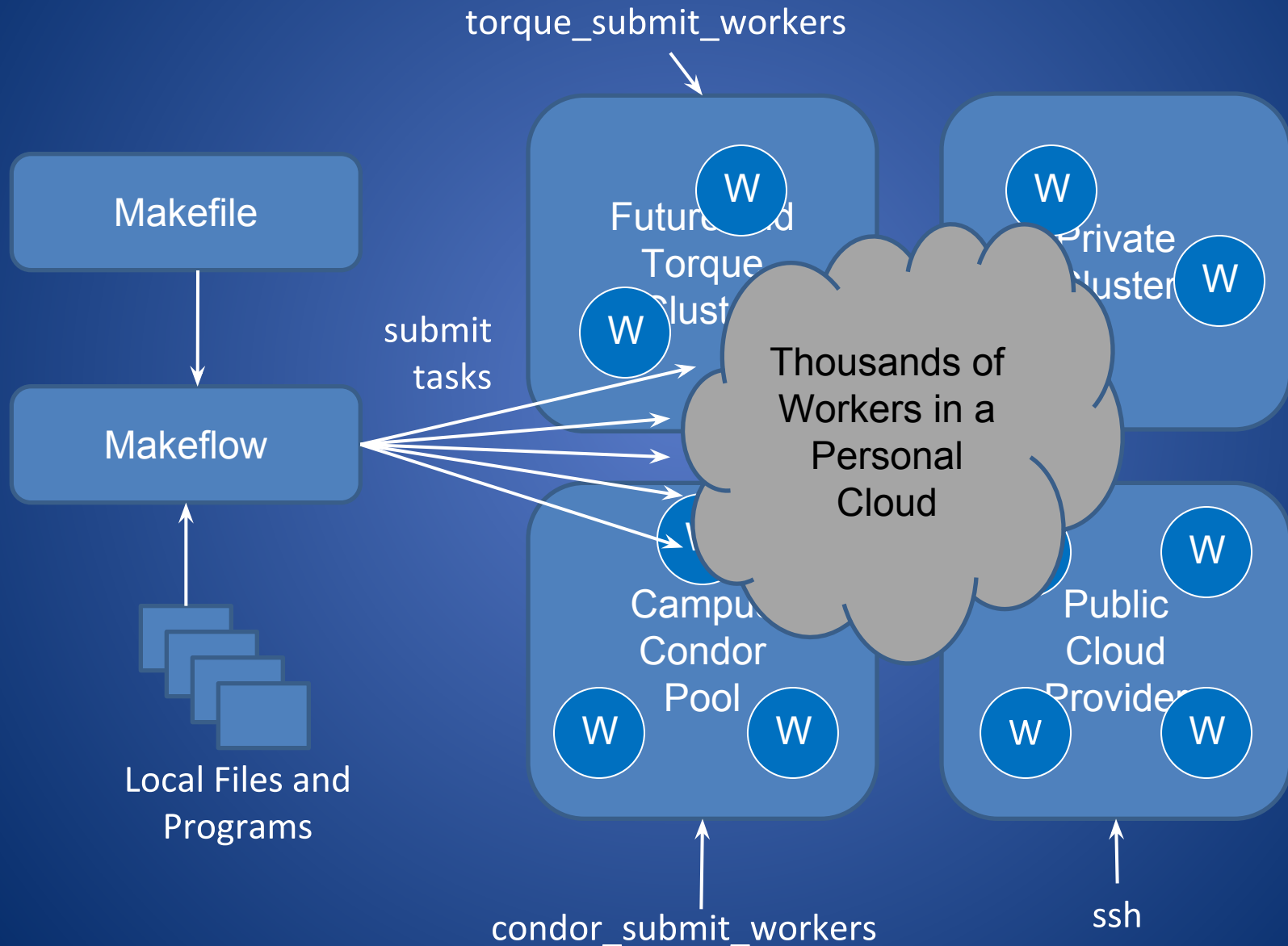
- Workflow Diagram (Left):** A Makeflow graph showing the flow of data. It starts with input files like 'serials_listig.hbk.sfx' and 'serials_listig.hbk.seq'. These feed into a 'blast' task, which produces 'output.1' through 'output.5' and 'error.1' through 'error.5'. These outputs are then collected and processed by 'collect', 'output', 'error', and 'total' tasks, leading to a 'finalize' task and finally a 'complete' state.
- BioCompute Web Interface (Top Right):** A screenshot of the BioCompute portal. The 'My Data' section shows a folder 'athrash1' (21.69 GB) containing files like 'nigenes.f.', '.fasta', 'nigenes.f.', 'CLIPPED-S.', 'SCRIPTS.A.', and 'CLIPPED.S.'. The 'Action' section shows 'Step 1 - Select Input File' and 'Step 2 - Title, Algorithm, and Privacy'. The 'My Queue' section shows a list of completed jobs with columns for Title, Status, and Username.
- File Browser (Bottom Right):** A screenshot of a web browser showing BLAST search results. The page is titled 'Biocompute @ Notre Dame - File Browser: 'ND5EP''. It displays search results for three queries:
 - Query: 152 FLAGFYSKDLELLEMLSLNLRNMFIFLFFVSTGLMFYIIR-LLMYMINDYN---LLII 207
 - Query: 208 YNLYD---ENYTLKSNFILLNMSVITGMSLWFIYSFYIPLNLRMLVLYVSMGL 264
 - Query: 265 LMGVLISNRMKIYSLNRPK 283
 The results show scores, identities, positives, and gaps for each query.

Makeflow + Work Queue

Makeflow + Batch System



Makeflow + Work Queue



Advantages of Work Queue

- Harness multiple resources simultaneously.
- Hold on to cluster nodes to execute multiple tasks rapidly. (ms/task instead of min/task)
- Scale resources up and down as needed.
- Better management of data, with local caching for data intensive tasks.
- Matching of tasks to nodes with data.

Makeflow and Work Queue

To start the Makeflow

```
% makeflow -T wq sims.mf
```

Could not create work queue on port 9123.

```
% makeflow -T wq -p 0 sims.mf
```

Listening for workers on port 8374...

To start one worker:

```
% work_queue_worker studentXX.cse.nd.edu 8374
```

Start Workers Everywhere

Submit workers to Condor:

```
condor_submit_workers studentXX.cse.nd.edu 8374 25
```

Submit workers to SGE:

```
sge_submit_workers studentXX.cse.nd.edu 8374 25
```

Submit workers to Torque:

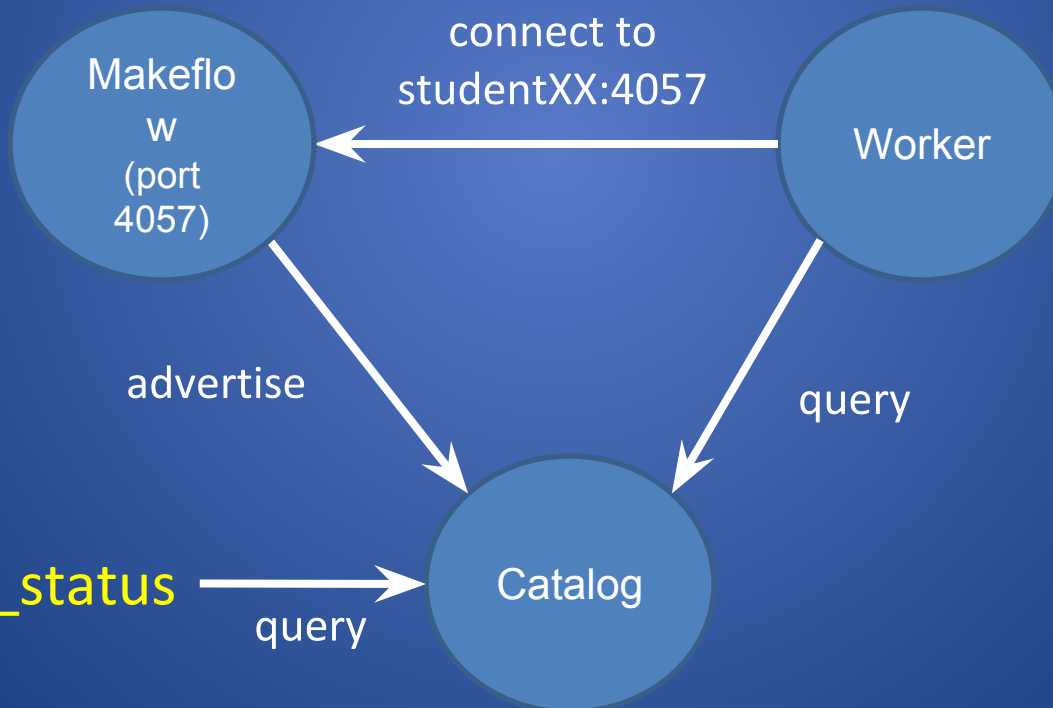
```
torque_submit_workers studentXX.cse.nd.edu 8374 25
```

Keeping track of port numbers
gets old fast...

Project Names

makeflow ...
-N myproject

work_queue_worker
-N myproject



work_queue_status → query

“myproject”
is at studentXX:4057

Project Names

Start Makeflow with a project name:

```
% makeflow -T wq -N myproject sims.mf
```

Listening for workers on port XYZ...

Start one worker:

```
% work_queue_worker -N myproject
```

Start many workers:

```
% torque_submit_workers -N myproject 5
```

work_queue_status

```
wizard.cse.nd.edu - PuTTY
% ./work_queue_status
PROJECT          NAME                PORT  WAITING  BUSY  COMPLETE  WORKERS
awe-fip35        fahnd04.crc.nd.edu  1024   719     1882  1206967   1882
hfeng-gromacs-10ps lclsstor01.crc.nd.edu 1024  4980     0    1280240   111
hfeng2-ala5      lclsstor01.crc.nd.edu 1025  2404    140   1234514   140
forcebalance     leeping.Stanford.EDU  5817  1082     26    822       26
forcebalance     leeping.Stanford.EDU  9230   0         3    147       3
fg-tutorial      login1.futuregrid.tacc 1024   3         0     0         0
% █
```

Resilience and Fault Tolerance

- MF +WQ is fault tolerant in many different ways:
 - If Makeflow crashes (or is killed) at any point, it will recover by reading the transaction log and continue where it left off.
 - Makeflow keeps statistics on both network and task performance, so that excessively bad workers are avoided.
 - If a worker crashes, the master will detect the failure and restart the task elsewhere.
 - Workers can be added and removed at any time during the execution of the workflow.
 - Multiple masters with the same project name can be added and removed while the workers remain.
 - If the worker sits idle for too long (default 15m) it will exit, so as not to hold resources idle.

Elastic Application Stack

All-Pairs

Wavefront

Makeflow

Custom
Apps

Work Queue Library

Thousands of Workers in a
Personal Cloud

W

W

Cam
Condu
Wol

W

Private
Cluste

W

W

Public
Cloud
Provider

W

W

Shared
SGE
Inst

W

W

Makeflow is an example of the
Directed Acyclic Graph
(or **Workflow**)
model of programming.

Work Queue is an example of the
Submit-Wait
model of programming.

(more on that next time)

The **Directed Acyclic Graph** model

can be implemented using

the **Submit-Wait** model