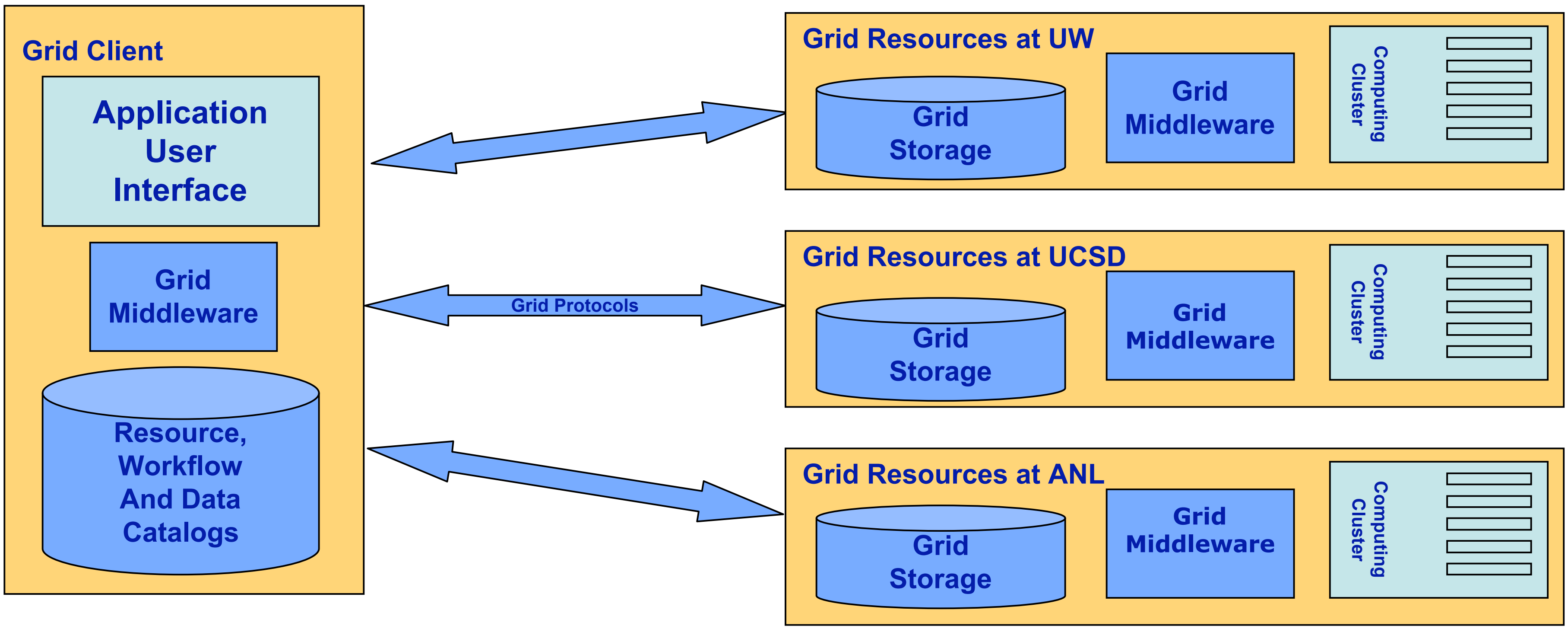
End User Tools

Target environment: Cluster and Grids
(distributed sets of clusters)



Running a uniform middleware stack:

- Security to control access and protect communication (GSI)
- Directory to locate grid sites and services: (VORS, MDS)
- Uniform interface to computing sites (GRAM)
- Facility to maintain and schedule queues of work (Condor-G)
- Fast and secure data set mover (GridFTP, RFT)
- Directory to track where datasets live (RLS)

High level tools for grid environments:

Workflow Management Systems

Why Workflow systems ?

- Advances in e-Sciences
  - Increasing amount of scientific datasets
- Growing complexity of scientific analyses
  - Procedures, algorythms
- 4 essential aspect of scientific computing addressed by workflows:
  - Describing complex scientific procedures
  - Automatic data derivation processes
  - High performance computing to improve throughput and performance
  - Provenance management and query

Design considerations

- New multi-core architecture -> radical changes in software design and development
  - Concurrency ?
    - How to write programs to take advantage of new architecture (greater computing and storage resources)

Scientific workflow systems

- Examples:
  - DAGMan
    - Provides a workflow engine that manages Condor jobs organized as DAGs (representing task precedence relationships)
    - Focus on scheduling and execution of long running jobs
  - Pegasus

Pegasus:

- Abstract Workflows - Pegasus input workflow description
  - workflow “high-level language”
  - only identifies the **computations** that a user wants to do
  - devoid of <u>resource descriptions</u>
  - devoid of <u>data locations</u>
- Pegasus
  - a workflow “compiler”
  - target language - DAGMan’s DAG and Condor submit files
  - transforms the workflow for performance and reliability
  - automatically locates physical locations for both workflow components and data
  - finds appropriate resources to execute the components
  - provides runtime provenance

Swift

- *Parallel scripting for distributed systems*

Authors:
Mike Wilde
wilde@mcs.anl.gov
Ben Clifford
benc@ci.uchicago.edu

- www.ci.uchicago.edu/swift

Why script in Swift?

- Orchestration of *many* resources over long time periods
  - Very complex to do manually - workflow automates this effort
- Enables restart of long running scripts
- Write scripts in a manner that's location-independent: run anywhere
  - *Higher level of abstraction* gives increased portability of the workflow script (over ad-hoc scripting)

Swift is…

- A language for writing scripts that:
  - process and produce large collections of data
  - with large and/or complex sequences of application programs
  - on diverse distributed systems
  - with a high degree of parallelism
  - persisting over long periods of time
  - surviving infrastructure failures
  - and tracking the provenance of execution

Swift programs

- A Swift script is a set of **functions**
  - Atomic functions wrap & invoke application programs
  - Composite functions invoke other functions
- Collections of **persistent file structures** (datasets) are **mapped** into this data model
- *Members of datasets can be processed in* ***parallel***
- Statements in a procedure are executed in **data-flow** dependency order and *concurrency*
- **Provenance** is gathered as scripts execute

A simple Swift script

```
type imagefile;

(imagefile output) flip(imagefile input) {
  app {
      convert "-rotate" "180" @input @output;
  }
}

imagefile stars <"orion.2008.0117.jpg">;
imagefile flipped <"output.jpg">;

flipped = flip(stars);
```

Parallelism via foreach { }

```
type imagefile;

(imagefile output) flip(imagefile input) {
  app {
    convert "-rotate" "180" @input @output;
  }
}

imagefile observations[ ] <simple_mapper; prefix=“orion”>;
imagefile flipped[ ]         <simple_mapper; prefix=“orion-flipped”>;
```

Name outputs based on inputs

```
foreach obs,i in observations {
  flipped[i] = flip(obs);
}
```

Process all dataset members in parallel

A Swift data mining example

```
type pcapfile;                    // packet data capture - input file type
type angleout;                    // “angle” data mining output
type anglecenter;   // geospatial centroid output

(angleout ofile, anglecenter cfile) angle4 (pcapfile ifile)
{
  app { angle4.sh --input @ifile --output @ofile --coords @cfile; }
// interface to shell script
}

pcapfile infile <"anl2-1182-dump.1.980.pcap">;    // maps real file

angleout      outdata <"data.out">;
anglecenter outcenter <"data.center">;

(outdata, outcenter) = angle4(infile);
```

Parallelism and name mapping

```
type pcapfile;
type angleout;
type anglecenter;

(angleout ofile, anglecenter cfile) angle4 (pcapfile ifile)
{
  app { angle4.sh --input @ifile --output @ofile --coords @cfile; }
}

pcapfile pcapfiles[]<filesys_mapper; prefix="pc", suffix=".pcap">;

angleout    of[] <structured_regexp_mapper;
  source=pcapfiles,match="pc(.*)\.pcap",
  transform="_output/of/of\1.angle">;

anglecenter cf[] <structured_regexp_mapper;
  source=pcapfiles,match="pc(.*)\.pcap",
  transform="_output/cf/cf\1.center">;
```

Name outputs based on inputs

```
foreach pf,i in pcapfiles {
  (of[i],cf[i]) = angle4(pf);
}
```

Iterate over dataset members in parallel

The Swift Scripting Model

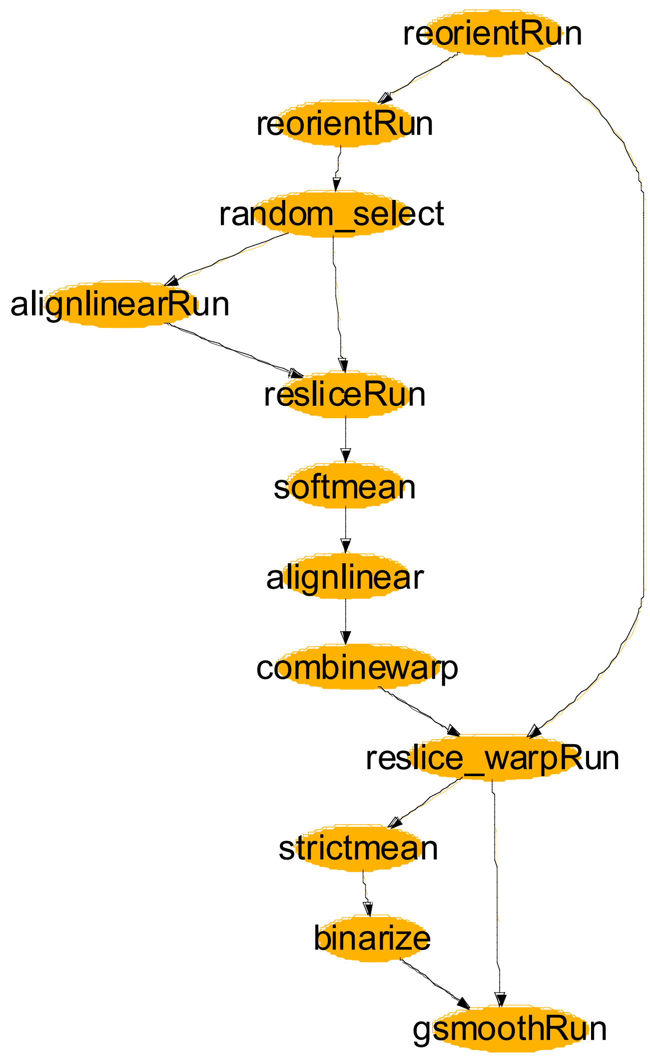
- Program in high-level, functional model
- Swift hides issues of *location*, *mechanism* and *data representation*
- Basic active elements are **functions** that encapsulate application tools and run jobs
- Typed data model:

  structures and arrays of

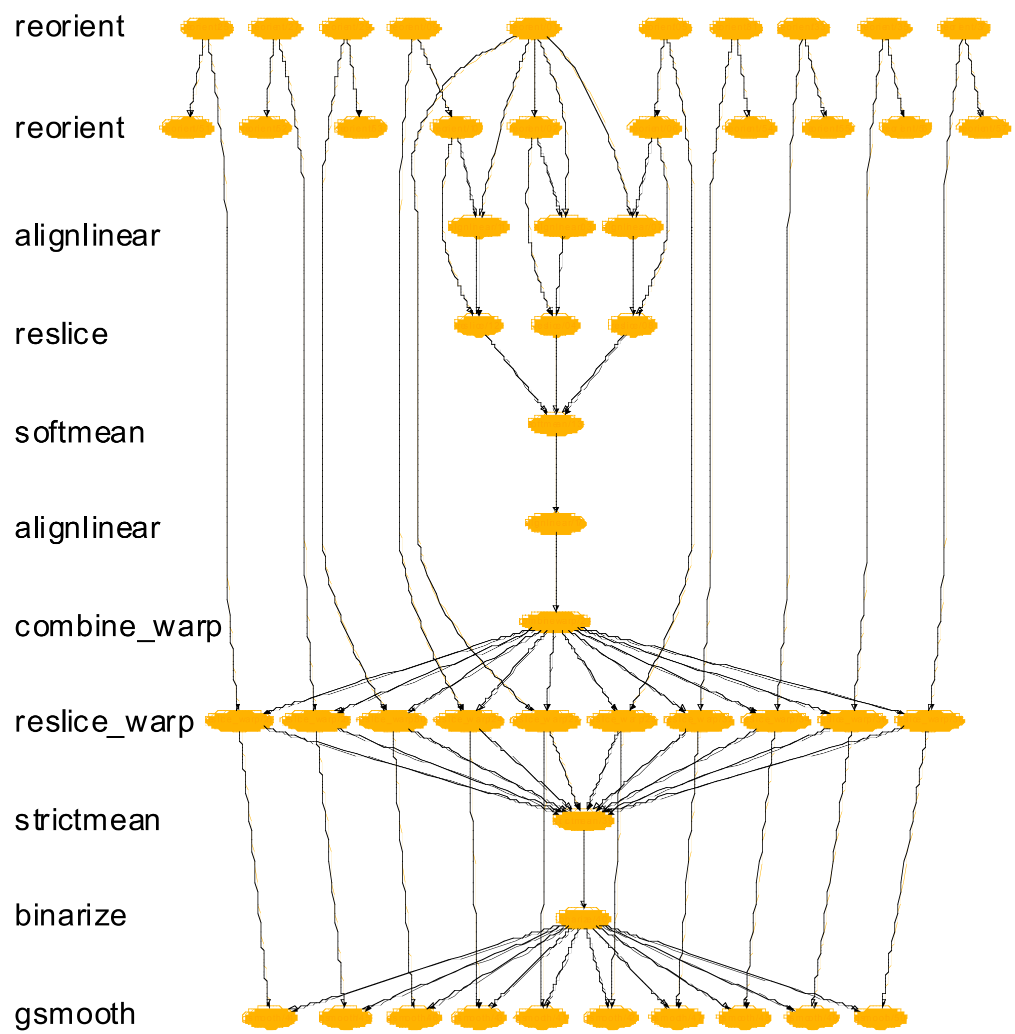
  files and scalar types
- Control structures perform conditional, iterative and *parallel* operations

Automated image registration for spatial normalization

AIRSN workflow:

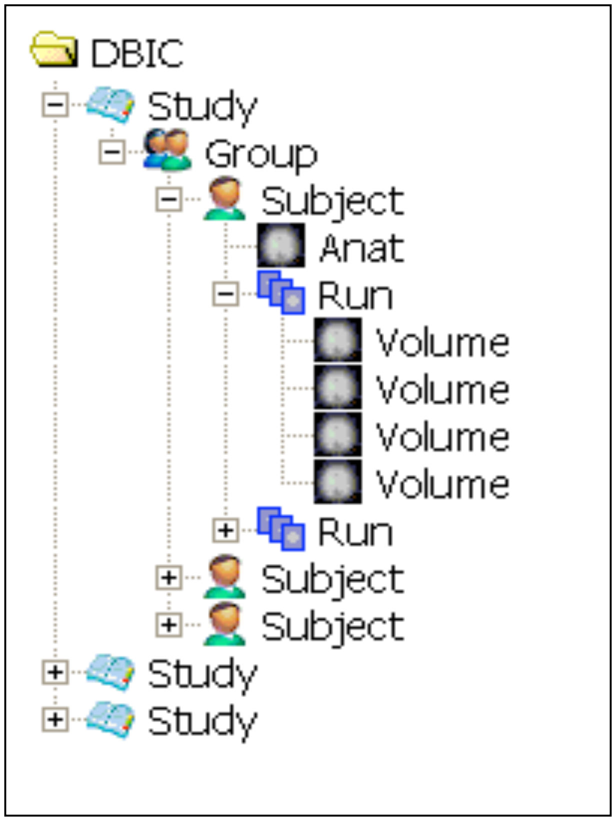


AIRSN workflow expanded:



Collaboration with James Dobson, Dartmouth [SIGMOD Record Sep05]

Example: fMRI Type Definitions



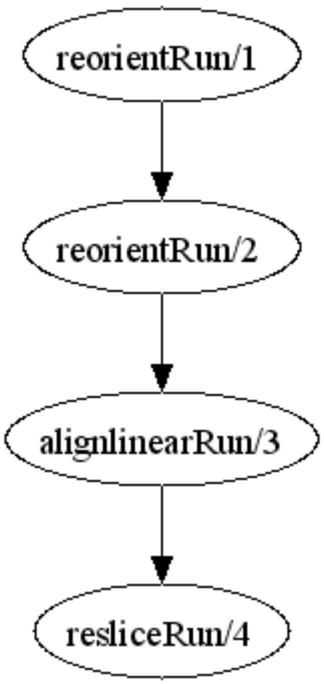
Simplified version of fMRI AIRSN Program (Spatial Normalization)

```
type Study {
        Group g[ ];
}

type Group {
        Subject s[ ];
}

type Subject {
        Volume anat;
        Run run[ ];
}

type Run {
        Volume v[ ];
}

type Volume {
        Image img;
        Header hdr;
}
```

```
type Image {};

type Header {};

type Warp {};

type Air {};

type AirVec {
        Air a[ ];
}

type NormAnat {
        Volume anat;
        Warp aWarp;
        Volume nHires;
}
```

fMRI Example Workflow

```
(Run resliced) reslice_wf ( Run r)
{
    Run yR = reorientRun( r , "y", "n" );
    Run roR  = reorientRun( yR , "x", "n" );
    Volume std = roR.v[1];
    AirVector roAirVec =
        alignlinearRun(std, roR, 12, 1000, 1000, "81 3 3");
    resliced = resliceRun( roR, roAirVec, "-o", "-k");
}
```




```
(Run or) reorientRun (Run ir, string direction, string overwrite)
{
      foreach Volume iv, i in ir.v {
            or.v[i] = reorient (iv, direction, overwrite);
      }
}
```

Collaboration with James Dobson, Dartmouth

Running swift

- Fully contained Java grid client
- Can test on a local machine
- Can run on a PBS cluster
- Runs on multiple clusters over Grid interfaces

Data Flow Model

- This is what makes it possible to be *location independent*
- Computations proceed when data is ready (often not in source-code order)
- User specifies DATA dependencies, doesn't worry about sequencing of operations
- Exposes maximal parallelism

Swift statements

- Var declarations
  - Can be mapped
- Type declarations
- Assignment statements
  - Assignments are type-checked
- Control-flow statements
  - if, foreach, iterate

- Function declarations

Swift: **Getting Started**

- www.ci.uchicago.edu/swift
  - Documentation -> tutorial
- Get CI accounts
  - https://www.ci.uchicago.edu/accounts/
    - Request: workstation, gridlab, teraport
- Get a DOEGrids Grid Certificate
  - http://www.doegrids.org/pages/cert-request.html
    - Virtual organization: OSG / OSGEDU
    - Sponsor: Mike Wilde, wilde@mcs.anl.gov, 630-252-7497
- Develop your Swift code and test locally, then:
  - On PBS / TeraPort
  - On OSG: OSGEDU
- Use simple scripts (Perl, Python) as your test apps

Swift: **Summary**

- Clean separation of logical/physical concerns
  - XDTM specification of logical data structures

+ Concise specification of parallel programs
  - SwiftScript, with iteration, etc.

+ Efficient execution (on distributed resources)
  - Grid interface, lightweight dispatch, pipelining, clustering

+ Rigorous provenance tracking and query
  - Records provenance data of each job executed

→ **Improved usability and productivity**
  - Demonstrated in numerous applications

Additional Information

- www.ci.uchicago.edu/swift
  - Quick Start Guide:
    - http://www.ci.uchicago.edu/swift/guides/quickstartguide.php
  - User Guide:
    - http://www.ci.uchicago.edu/swift/guides/userguide.php
  - Introductory Swift Tutorials:
    - http://www.ci.uchicago.edu/swift/docs/index.php

DOCK - example

- Molecular dynamics application example
- Use Swift

DOCK -steps

(0) Create valid proxy based on your certificate with ‘grid-proxy-init’.
We assume your cert is mapped to the OSG VO.

(1) Download and setup adem-osg toolkits
svn co https://svn.ci.uchicago.edu/svn/vdl2/SwiftApps/adem-osg adem-osg
(ADEM = Application Deployment and Management tool)

This set of scripts is used to automate the end user process. It deals with:

- the detecting the available OSG resources (creation of sites.xml file)
- creation of remote working directories on these sites on which authentication tests were successful
- creation of appropriate tc.data catalogs (that contain information about the sites and location of where DOCK application is installed)

This way, many of the grid related processing steps are hidden from the users and performed via the scripts provided.

(2) Get the available grid sites and sites.xml for swift

> *auto-get-sites $GRID $VO*

(get the available grid sites within a given virtual organization in osg or osg-itb) e.g. “auto-get-sites osg osg”

(3) prepare-for-dock-swift-submit

> *./prepare-for-dock-swift-submit $VO $Grid-sites-file*

(e.g. ./prepare-for-dock-swift-submit osg $ADEM_HOME/tmp/osg-osg-avail-sites-$DATE.txt)

(4) update .swift source file

(5) Submit the job

```
$ swift   -sites.file ../swift-sites.xml  -tc.file ./dock-tc.data
          grid-many-dock6-auto.swift
```

site	JOB_START	JOB_END	APPLICATION_EXCEPTION	JOB_CANCELED	unknow n	total
AGLT2	0	985	4	89	0	1078
CIT_CMS_T2	0	0	20	2	0	22
GLOW-CMS	0	1160	106	194	1	1461
NYSGRID-CCR-U2	0	841	1	335	0	1177
OSG_LIGO_MIT	0	877	1	106	0	984
SMU_PHY	0	522	0	37	0	559
TTU-ANTAEUS	0	168	1	122	1	292

Tools for graphical log processing