Introduction to Swift

Parallel scripting for distributed systems

Mike Wilde
wilde@mcs.anl.gov

Ben Clifford
benc@ci.uchicago.edu

Computation Institute, University of Chicago
and Argonne National Laboratory

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 persistent 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
- Data is **typed** as composable arrays and structures of files and simple scalar types (int, float, string)
- 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
- Variables are **single assignment**
- **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

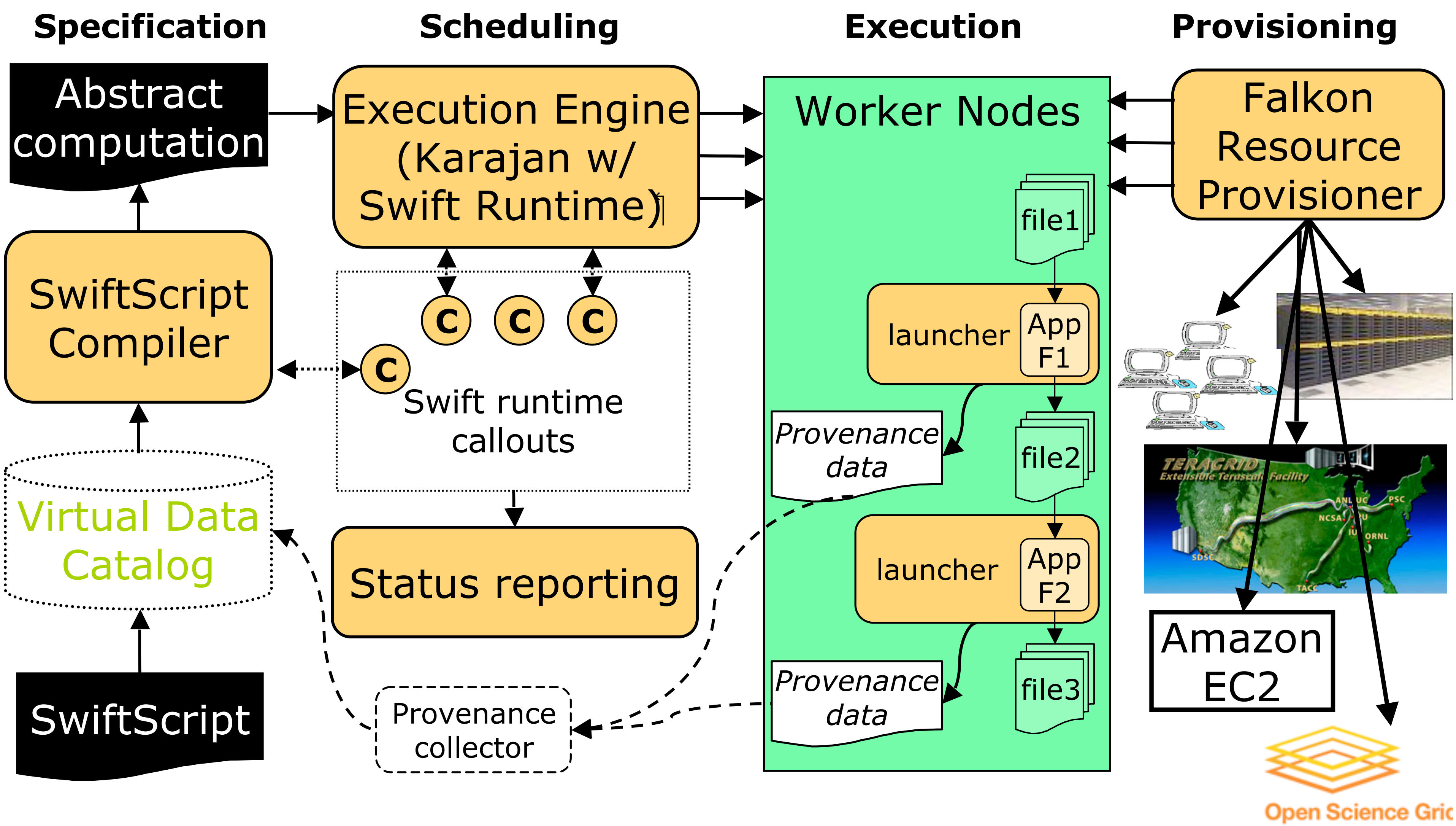
```
foreach pf,i in pcapfiles {
  (of[i],cf[i]) = angle4(pf);
}
```

Iterate over dataset members in parallel

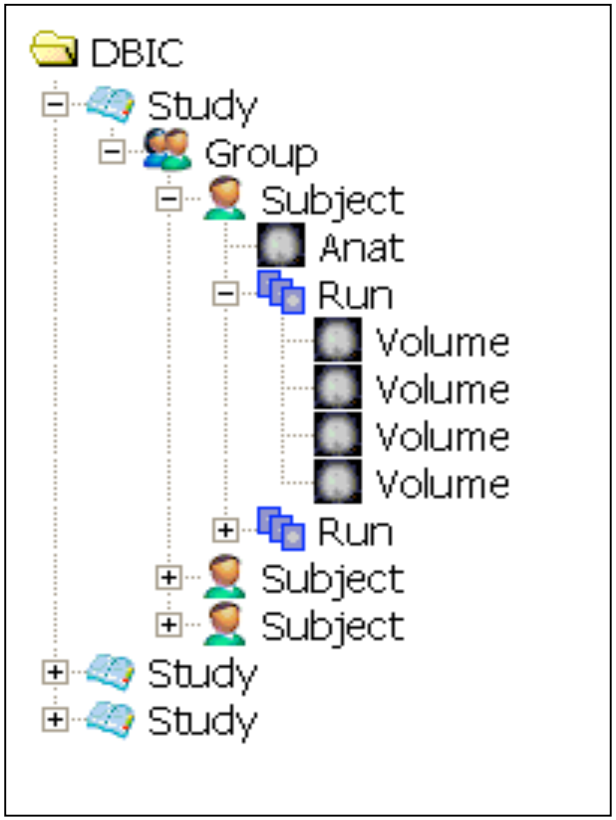
The Swift Scripting Model

- 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
- Variables are single assignment
- Control structures perform conditional, iterative and *parallel* operations

Swift Architecture



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** {};

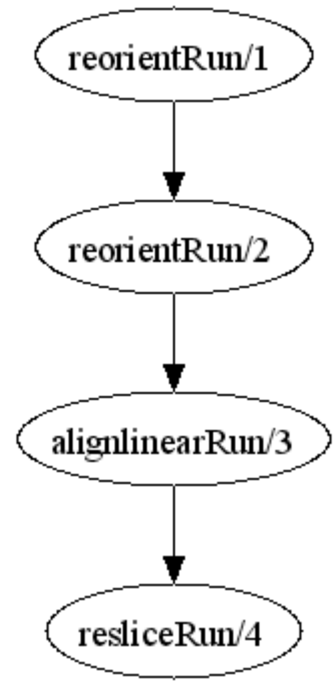
type **Air** {};

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

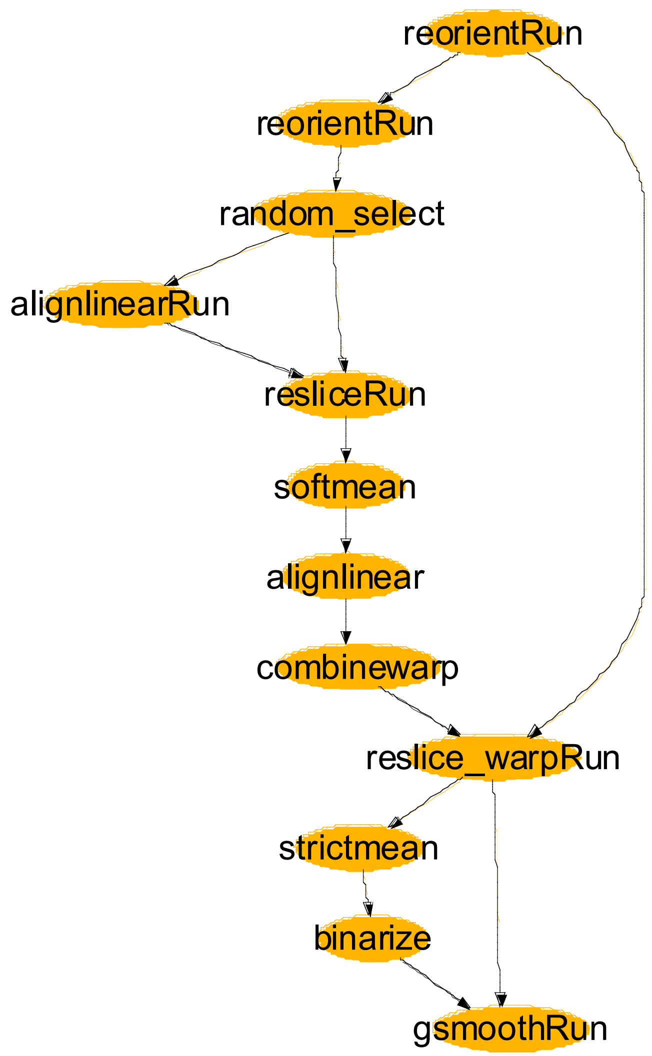
AIRSN Program Definition

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

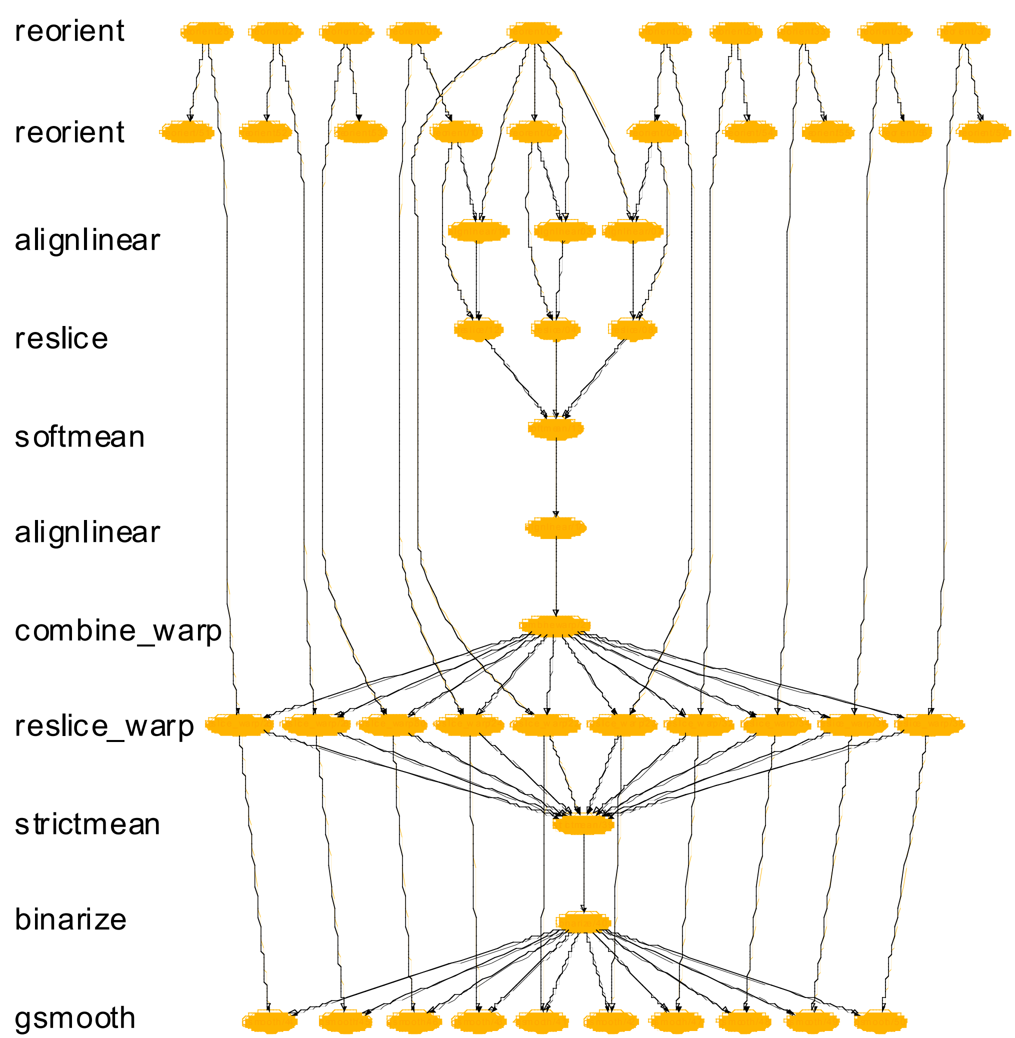
```
(Run snr) functional ( Run r, NormAnat a,
                       Air shrink ) {
    Run yroRun = reorientRun( r , "y" );
    Run roRun = reorientRun( yroRun , "x" );
    Volume std = roRun[0];
    Run rndr = random_select( roRun, 0.1 );
    AirVector rndAirVec = align_linearRun( rndr, std, 12, 1000, 1000, "81 3 3" );
    Run reslicedRndr = resliceRun( rndr, rndAirVec, "o", "k" );
    Volume meanRand = softmean( reslicedRndr, "y", "null" );
    Air mnQAAir = alignlinear( a.nHires, meanRand, 6, 1000, 4, "81 3 3" );
    Warp boldNormWarp = combinewarp( shrink, a.aWarp, mnQAAir );
    Run nr = reslice_warp_run( boldNormWarp, roRun );
    Volume meanAll = strictmean( nr, "y", "null" )
    Volume boldMask = binarize( meanAll, "y" );
    snr = gsmoothRun( nr, boldMask, "6 6 6" );
}
```

Automated image registration for spatial normalization

AIRSN workflow:



AIRSN workflow expanded:



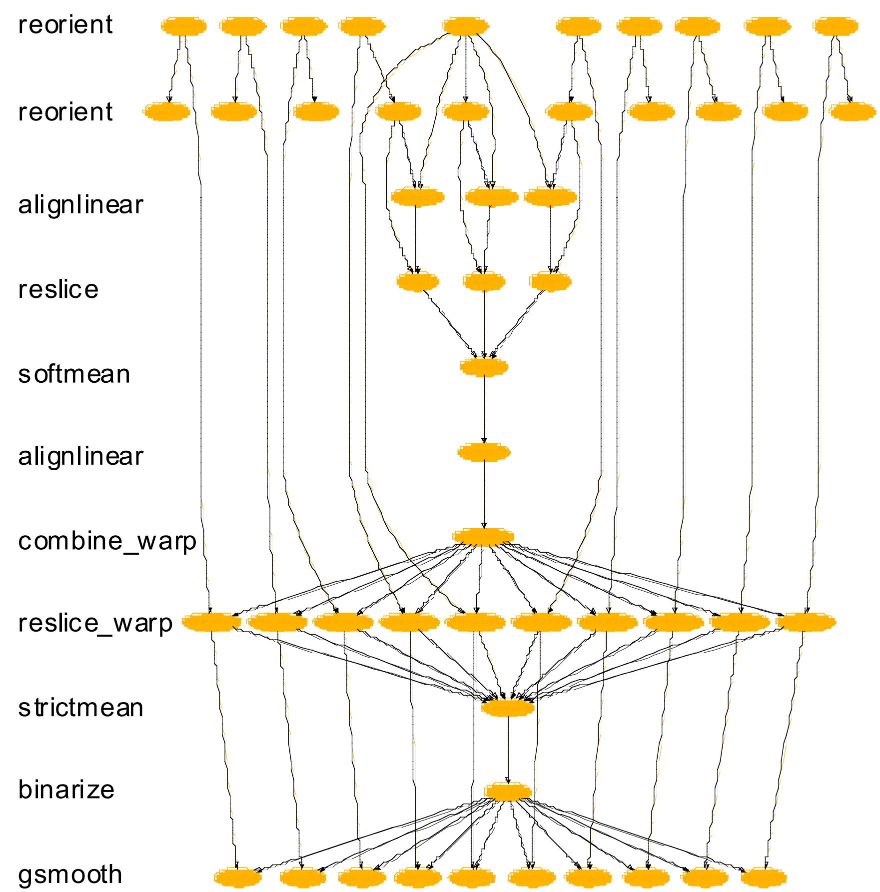
Collaboration with James Dobson, Dartmouth [SIGMOD Record Sep05]

SwiftScript Expressiveness

Lines of code with different workflow encodings

fMRI Workflow	*Shell Script*	*VDL*	*Swift*
ATLAS1	**49**	**72**	**6**
ATLAS2	**97**	**135**	**10**
FILM1	**63**	**134**	**17**
FEAT	**84**	**191**	**13**
AIRSN	**215**	**~400**	**34**

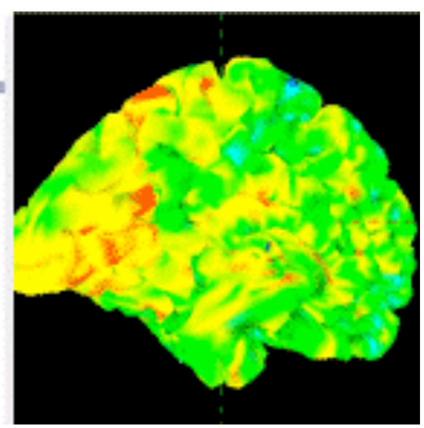
AIRSN workflow:expanded:



Collaboration with James Dobson, Dartmouth [SIGMOD Record Sep05]

Application example: ACTIVAL: Neural activation validation

The ACTIVAL Swift script identifies clusters of neural activity not likely to be active by random chance: switch labels of the conditions for one or more participants; calculate the delta values in each voxel, re-calculate the reliability of delta in each voxel, and evaluate clusters found. If the clusters in data are greater than the majority of the clusters found in the permutations, then the null hypothesis is refuted indicating that clusters of activity found in our experiment are not likely to be found by chance.



Work by S. Small and U. Hasson, UChicago.

SwiftScript Workflow ACTIVAL – Data types and utilities

```
type script {}                          type fullBrainData {}
type brainMeasurements{}                type fullBrainSpecs {}
type precomputedPermutations{}          type brainDataset {}
type brainClusterTable {}
type brainDatasets{ brainDataset b[]; }
type brainClusters{ brainClusterTable c[]; }
```

```
// Procedure to run "R" statistical package
(brainDataset t) bricRInvoke (script permutationScript, int iterationNo,
    brainMeasurements dataAll, precomputedPermutations dataPerm) {
      app { bricRInvoke @filename(permutationScript) iterationNo
                    @filename(dataAll) @filename(dataPerm); }
}
```

```
// Procedure to run AFNI Clustering tool
(brainClusterTable v, brainDataset t) bricCluster (script clusterScript,
  int iterationNo, brainDataset randBrain, fullBrainData brainFile,
  fullBrainSpecs specFile) {
      app { bricPerlCluster @filename(clusterScript) iterationNo
                    @filename(randBrain) @filename(brainFile)
                    @filename(specFile); }
}
```

```
// Procedure to merge results based on statistical likelhoods
(brainClusterTable t) bricCentralize ( brainClusterTable bc[]) {
     app { bricCentralize @filenames(bc); }
}
```

ACTIVAL Workflow – Dataset iteration procedures

```
// Procedure to iterate over the data collection

(brainClusters randCluster, brainDatasets dsetReturn) brain_cluster
  (fullBrainData brainFile, fullBrainSpecs specFile)
{
  int sequence[]=[1:2000];

  brainMeasurements          dataAll<fixed_mapper; file="obs.imit.all">;
  precomputedPermutations dataPerm<fixed_mapper; file="perm.matrix.11">;
  script                     randScript<fixed_mapper; file="script.obs.imit.tibi">;
  script                     clusterScript<fixed_mapper; file="surfclust.tibi">;
  brainDatasets              randBrains<simple_mapper; prefix="rand.brain.set">;

  foreach int i in sequence {
     randBrains.b[i] = bricRInvoke(randScript,i,dataAll,dataPerm);
     brainDataset rBrain = randBrains.b[i] ;
     (randCluster.c[i],dsetReturn.b[i]) =
        bricCluster(clusterScript,i,rBrain, brainFile,specFile);
  }
}
```

ACTIVAL Workflow – Main Workflow Program

// Declare datasets

```
fullBrainData        brainFile<fixed_mapper; file="colin_lh_mesh140_std.pial.asc">;
fullBrainSpecs       specFile<fixed_mapper; file="colin_lh_mesh140_std.spec">;

brainDatasets        randBrain<simple_mapper; prefix="rand.brain.set">;
brainClusters        randCluster<simple_mapper; prefix="Tmean.4mm.perm",
                          suffix="_ClstTable_r4.1_a2.0.1D">;
brainDatasets        dsetReturn<simple_mapper; prefix="Tmean.4mm.perm",
                          suffix="_Clustered_r4.1_a2.0.niml.dset">;
brainClusterTable    clusterThresholdsTable<fixed_mapper; file="thresholds.table">;
brainDataset         brainResult<fixed_mapper; file="brain.final.dset">;
brainDataset         origBrain<fixed_mapper; file="brain.permutation.1">;
```

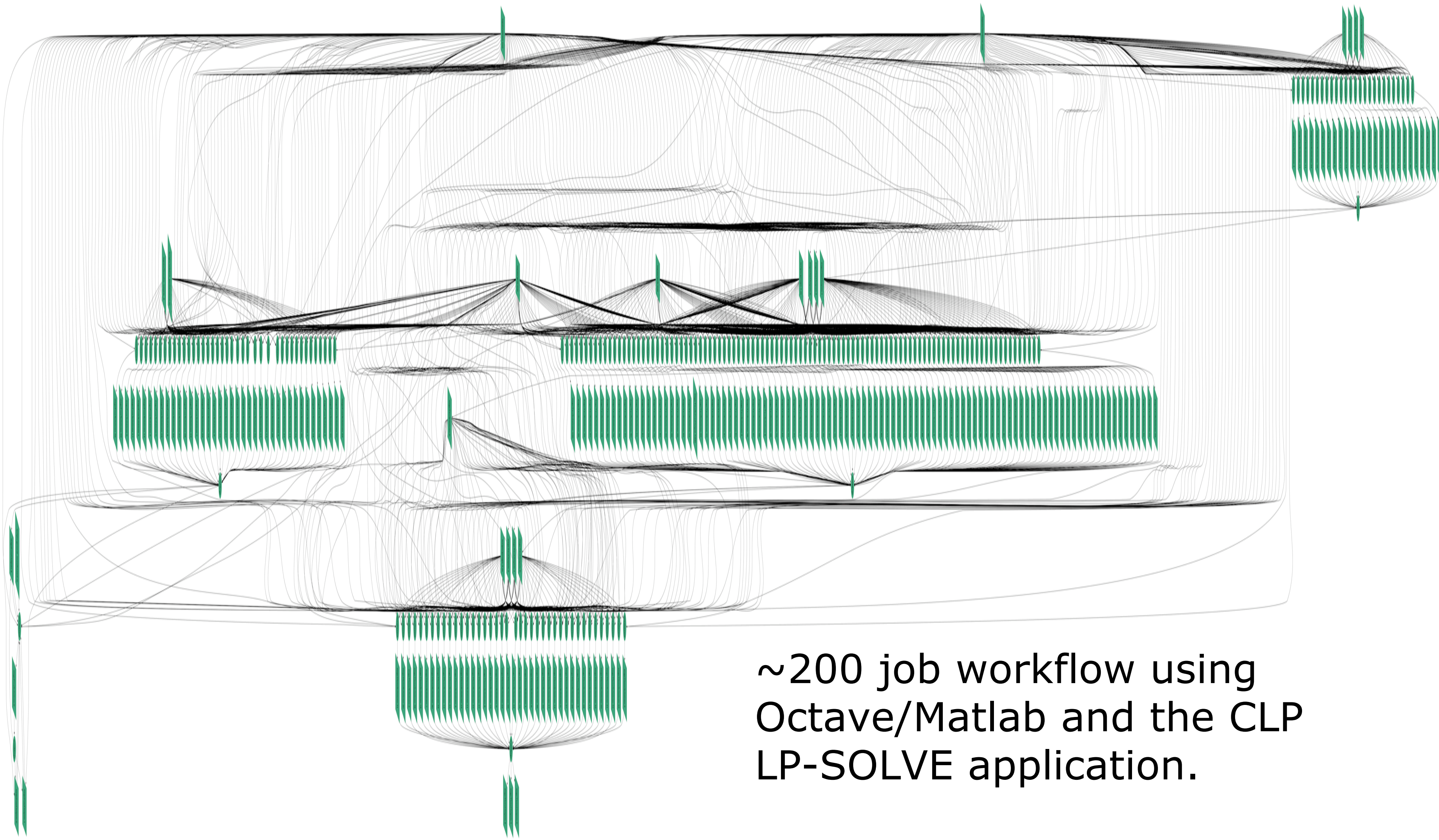
// Main program – executes the entire workflow

```
(randCluster, dsetReturn) = brain_cluster(brainFile, specFile);

clusterThresholdsTable = bricCentralize (randCluster.c);

brainResult = makebrain(origBrain,clusterThresholdsTable,brainFile,specFile);
```

Swift Application: Economics "moral hazard" problem



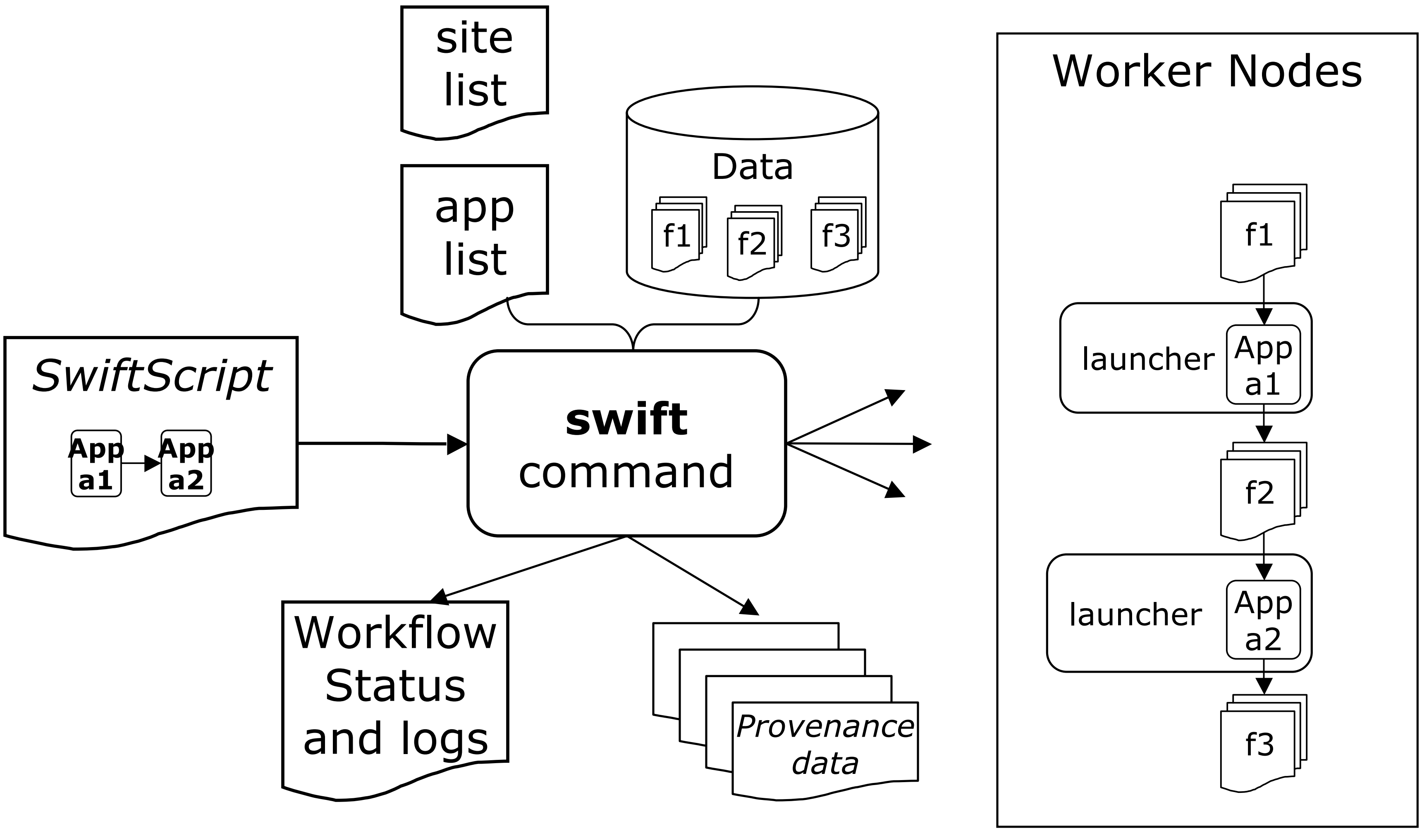
~200 job workflow using Octave/Matlab and the CLP LP-SOLVE application.

Work by Tibi Stef-Praun, CI, with Robert Townsend & Gabriel Madiera, UChicago Economics

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

Using Swift


site
list
app
list
Data
f1
f2
f3
Worker Nodes
f1
launcher
App
a1
f2
launcher
App
a2
f3
SwiftScript
App
a1
App
a2
swift
command
Workflow
Status
and logs
Provenance
data

The Variable model

- Single assignment:
  – Can only assign a value to a var once
  – This makes data flow semantics much cleaner to specify, understand and implement
- Variables are scalars or references to composite objects
- Variables are typed
- File typed variables are “mapped” to files

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

Passing scripts as data

- When running scripting languages, target language interpreter can be the executable
- Powerful technique for running scripts in:
  - sh, bash
  - Perl, Python, Tcl
  - R, Octave
- These are often pre-installed at known places
  - No application installation needed
- Need to deal with library modules manually

Assessing your analysis tool performance

- Job usage records tell where when and how things ran: V runtime cputime

angle4-szlfhtji-kickstart.xml 2007-11-08T23:03:53.733-06:00 0 0 1177.024 1732.503 4.528 ia64 tg-c007.uc.teragrid.org

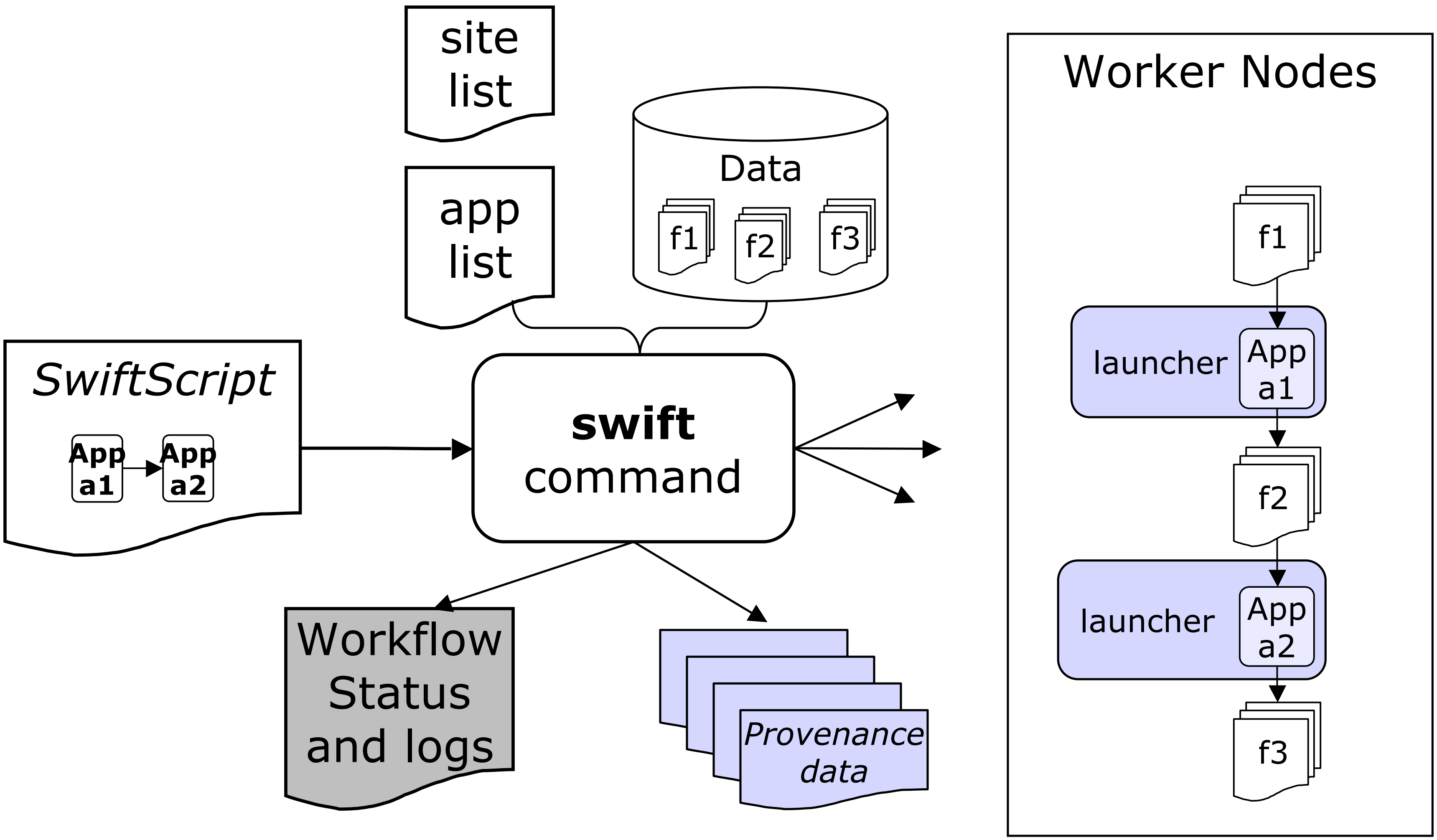
angle4-hvlfhtji-kickstart.xml 2007-11-08T23:00:53.395-06:00 0 0 1017.651 1536.020 4.283 ia64 tg-c034.uc.teragrid.org

angle4-oimfhtji-kickstart.xml 2007-11-08T23:30:06.839-06:00 0 0 868.372 1250.523 3.049 ia64 tg-c015.uc.teragrid.org

angle4-u9mfhtji-kickstart.xml 2007-11-08T23:15:55.949-06:00 0 0 817.826 898.716 5.474 ia64 tg-c035.uc.teragrid.org

- Analysis tools display this visually

Performance recording



Data Management

- Directories and management model
  - local dir, storage dir, work dir
  - caching within workflow
  - reuse of files on restart
- Makes unique names for: jobs, files, wf
- Can leave data on a site
  - For now, in Swift you need to track it
  - In Pegasus (and VDS) this is done automatically

Mappers and Vars

- Vars can be “file valued”
- Many useful mappers built-in, written in Java to the Mapper interface
- “Ext”ernal mapper can be easily written as an external script in any language

Mapping outputs based on input names

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

(angleout ofile, anglecenter cfile) angle4 (pcapfile ifile)
{
  app { angle4 @ifile @ofile @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">;

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

Name outputs based on inputs

Parallelism for processing datasets

```
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

Coding your own “external” mapper

```
awk <angle-spool-1-2 '
BEGIN {
  server="gsiftp://tp-osg.ci.uchicago.edu//disks/ci-gpfs/angle";
 }
{   printf "[%d] %s/%s\n", i++, server, $0    }’
```

```
$ cat angle-spool-1-2
spool_1/anl2-1182294000-dump.1.167.pcap.gz
spool_1/anl2-1182295800-dump.1.170.pcap.gz
spool_1/anl2-1182296400-dump.1.171.pcap.gz
spool_1/anl2-1182297600-dump.1.173.pcap.gz
…
$ ./map1 | head
[0] gsiftp://tp-osg.ci.uchicago.edu//disks/ci-gpfs/angle/spool_1/anl2-1182294000-
    dump.1.167.pcap.gz
[1] gsiftp://tp-osg.ci.uchicago.edu//disks/ci-gpfs/angle/spool_1/anl2-1182295800-
    dump.1.170.pcap.gz
[2] gsiftp://tp-osg.ci.uchicago.edu//disks/ci-gpfs/angle/spool_1/anl2-1182296400-
    dump.1.171.pcap.gz
    …
```

Site selection and throttling

- Avoid overloading target infrastructure
- Base resource choice on current conditions and real response for *you*
- Balance this with space availability
- Things are getting more automated.

Clustering and Provisioning

- Can cluster jobs together to reduce grid overhead for small jobs
- Can use a provisioner
- Can use a provider to go straight to a cluster

Testing and debugging techniques

- Debugging
  - Trace and print statements
  - Put logging into your wrapper
  - Capture stdout/error in returned files
  - Capture glimpses of runtime environment
  - Kickstart data helps understand what happened at runtime
  - Reading/filtering swift client log files
  - Check what sites are doing with local tools - condor_q, qstat
- Log reduction tools tell you how your workflow behaved

Other Workflow Style Issues

- Expose or hide parameters
- One atomic, many variants
- Expose or hide program structure
- Driving a parameter sweep with readdata() - reads a csv file into struct[].
- Swift is not a data manipulation language - use scripting tools for that

Swift: Getting Started

- www.ci.uchicago.edu/swift
  - Documentation -> tutorials
- 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

Planned Enhancements

- Additional data management models
- Integration of provenance tracking
- Improved logging for troubleshooting
- Improved compilation and tool integration (especially with scripting tools and SDEs)
- Improved abstraction and descriptive capability in mappers
- Continual performance measurement and speed improvement

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)
  - **Karajan+Falkon**: Grid interface, lightweight dispatch, pipelining, clustering, provisioning

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

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

http://www.ci.uchicago.edu/swift

Acknowledgments

- Swift effort is supported in part by NSF grants OCI-721939 and PHY-636265, NIH DC08638, and the UChicago/Argonne Computation Institute
- The Swift team:
  - Ben Clifford, Ian Foster, Mihael Hategan, Veronika Nefedova, Ioan Raicu, Tibi Stef-Praun, Mike Wilde, Zhao Zhang, Yong Zhao
- Java CoG Kit used by Swift developed by:
  - Mihael Hategan, Gregor Von Laszewski, and many collaborators
- User contributed workflows and application use
  - I2U2, U.Chicago Molecular Dynamics, U.Chicago Radiology and Human Neuroscience Lab, Dartmouth Brain Imaging Center

References - Workflow

- Taylor, I.J., Deelman, E., Gannon, D.B. and Shields, M. eds. Workflows for e-Science, Springer, 2007
- SIGMOD Record Sep. 2005 Special Section on Scientific Workflows, http://www.sigmod.org/sigmod/record/issues/0509/index.html
- Zhao Y., Hategan, M., Clifford, B., Foster, I., vonLaszewski, G., Raicu, I., Stef-Praun, T. and Wilde, M Swift: Fast, Reliable, Loosely Coupled Parallel Computation IEEE International Workshop on Scientific Workflows 2007
- Stef-Praun, T., Clifford, B., Foster, I., Hasson, U., Hategan, M., Small, S., Wilde, M and Zhao,Y. Accelerating Medical Research using the Swift Workflow System Health Grid 2007
- Stef-Praun, T., Madeira, G., Foster, I., and Townsend, R. Accelerating solution of a moral hazard problem with Swift e-Social Science 2007
- Zhao, Y., Wilde, M. and Foster, I. Virtual Data Language: A Typed Workflow Notation for Diversely Structured Scientific Data. Taylor, I.J., Deelman, E., Gannon, D.B. and Shields, M. eds. Workflows for eScience, Springer, 2007, 258-278.
- Zhao, Y., Dobson, J., Foster, I., Moreau, L. and Wilde, M. A Notation and System for Expressing and Executing Cleanly Typed Workflows on Messy Scientific Data. SIGMOD Record 34 (3), 37-43
- Vöckler, J.-S., Mehta, G., Zhao, Y., Deelman, E. and Wilde, M., Kickstarting Remote Applications. 2nd International Workshop on Grid Computing Environments, 2006.
- Raicu, I., Zhao Y., Dumitrescu, C., Foster, I. and Wilde, M Falkon: a Fast and Light-weight tasK executiON framework Supercomputing Conference 2007

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