Swift
Fast, Reliable, Loosely Coupled Parallel Computation

Ian Foster
Computation Institute
Argonne National Laboratory
University of Chicago

Joint work with Yong Zhao, Ioan Raicu, Mike Wilde, Ben Clifford, Mihael Hatigan, Tibi Stef-Praun, Veronika Nefedova
Case Study: The Functional MRI (fMRI) Data Center

- Online repository of neuroimaging data
  - A typical study comprises 3 groups, 20 subjects/group, 5 runs/subject, 300 volumes/run
    → 90,000 volumes, 60 GB raw
    → 1.2 million files processed
  - 100s of such studies in total

- Many users analyze this data
  - Wide range of analyses
  - Testing → production
  - Ensembles: a set of data analyses by parameters, datasets
fMRI: A Broad Picture
Challenges

- Deluge of data: instrumentation, simulation
- Data analysis turns into data integration
- Community-wide collaboration
- Provenance: tracking, query, application
- Scalability: desktop to Grid
- Productivity: throughput, performance
**Swift System**

- 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** threading, **Falkon** provisioning, **Globus** interfaces, pipelining, load balancing
- Rigorous provenance tracking and query
  - Virtual data schema & automated recording

→ **Improved usability and productivity**
  - Demonstrated in numerous applications
The Messy Data Problem

- Scientific data is often logically structured
  - E.g., hierarchical structure
  - Common to map functions over dataset members
  - Nested map operations can scale to millions of objects
The Messy Data Problem

- But physically “messy”
- Heterogeneous storage format and access protocol
  - Logically identical dataset can be stored in textual File (e.g. CSV), spreadsheet, database, ...
  - Data available from filesystem, DBMS, HTTP, WebDAV, ..
- Metadata encoded in directory and file names
- Hinders program development, composition, execution

```
./knottastic
total 58
drwxr-xr-x 4 yongzh users 2048 Nov 12 14:15 AA
drwxr-xr-x 4 yongzh users 2048 Nov 11 21:13 CH
drwxr-xr-x 4 yongzh users 2048 Nov 11 16:32 EC

./knottastic/AA:
total 4
drwxr-xr-x 5 yongzh users 2048 Nov 5 12:41 04nov06aa
drwxr-xr-x 4 yongzh users 2048 Dec 6 12:24 11nov06aa

./knottastic//AA/04nov06aa:
total 54
drwxr-xr-x 2 yongzh users 2048 Nov 5 12:52 ANATOMY
drwxr-xr-x 2 yongzh users 49152 Dec 5 11:40 FUNCTIONAL

./knottastic/AA/04nov06aa/ANATOMY:
total 58500
-rw-r--r-- 1 yongzh users 348 Nov 5 12:29 coplanar.hdr
-rw-r--r-- 1 yongzh users 16777216 Nov 5 12:29 coplanar.img

./knottastic/AA/04nov06aa/FUNCTIONAL:
total 196739
-rw-r--r-- 1 yongzh users 348 Nov 5 12:32 bold1_0001.hdr
-rw-r--r-- 1 yongzh users 409600 Nov 5 12:32 bold1_0001.img
-rw-r--r-- 1 yongzh users 348 Nov 5 12:32 bold1_0002.hdr
-rw-r--r-- 1 yongzh users 409600 Nov 5 12:32 bold1_0002.img
-rw-r--r-- 1 yongzh users 496 Nov 15 20:44 bold1_0002.mat
-rw-r--r-- 1 yongzh users 348 Nov 5 12:32 bold1_0003.hdr
-rw-r--r-- 1 yongzh users 409600 Nov 5 12:32 bold1_0003.img
```
XML Dataset Typing & Mapping (XDTM)

- Describe logical structure by **XML Schema**
  - Primitive scalar types: int, float, string, date, ...
  - Complex types (structs and arrays)
- Use **mapping descriptors** for mappings
  - How dataset elements are mapped to physical representations
  - External parameters (e.g., location)
- Use **XPath** for dataset selection

XDTM: XML Dataset Typing and Mapping for Specifying Datasets [EGC05]
XDTM: Related Work

- **Data format standardization**
  - FITS, CDF, HDF-5, DICOM

- **Data format description**
  - DFDL [Beckerle, Westhead04] embeds annotations with XML Schema
  - PADS [Fisher, Gruber05], PADX [Fernandez, Fisher06], declarative specs of physical layout and semantic properties

- **Logical object**
  - ADO [Microsoft01], in memory relational model
  - SDO [Beatty, Brodsky03], logical data model for J2EE programming
XDTM: Implementation

- Virtual integration
  - Each data source treated as virtual XML source
  - Data structure defined as XML schema
  - Mapper responsible for accessing source and translating to/from XML representation
  - Bi-directional

- Common mapping interface
  - Data providers implement the interface
    - Responsible for data access details
  - Standard mapper implementations provided
    - String, file system, CSV, ...
SwiftScript

- **Typed parallel programming** [SIGMOD05, Springer06]
  - XDTM as data model and type system
  - Typed dataset and procedure definitions
- **Scripting language**
  - Implicit data parallelism
  - Program composition from procedures
  - Control constructs (foreach, if, while, ...)

Clean application logic
Type checking
Dataset selection, iteration
Discovery by types
Type conversion

A Notation & System for Expressing and Executing Cleanly Typed Workflows on Messy Scientific Data [SIGMOD05]
SwiftScript: Related Work

- **Coordination language**
  - Linda [Ahuja, Carriero86], Strand [Foster, Taylor90], PCN [Foster92]
  - Durra [Barbacci, Wing86], MANIFOLD [Papadopoulos98]
  - Components programmed in specific language (C, FORTRAN) and linked with system

- **“Workflow” languages and systems**
  - Taverna [Oinn, Addis04], Kepler [Ludäscher, Altintas05], Triana [Churches, Gombas05], Vistrail [Callahan, Freire06], DAGMan, Star-P
  - XPDL [WfMC02], BPEL [Andrews, Curbera03], and BPML [BPML02], YAWL [van de Aalst, Hofstede05], Windows Workflow Foundation [Microsoft05]
## Related Work

<table>
<thead>
<tr>
<th></th>
<th>SwiftScript</th>
<th>BPEL</th>
<th>XPDL</th>
<th>MW Wflow</th>
<th>DAGMan</th>
<th>Taverna</th>
<th>Triana</th>
<th>Kepler</th>
<th>Vistrail</th>
<th>Star-P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scales to Grids</td>
<td>++</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>++</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Typing</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Iteration</td>
<td>++</td>
<td>-/+</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Scripting</td>
<td>++</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>++</td>
</tr>
<tr>
<td>Dataset Mapping</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Service Interop</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Subflow/comp.</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Provenance</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Open source</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
</tbody>
</table>

“A 4x200 flow leads to a 5 MB BPEL file ... chemists were not able to write in BPEL” [Emmerich,Buchart06]
fMRI Type Definitions in SwiftScript

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;
}
Type Definitions in XML Schema

```xml
<xs:schema targetNamespace="http://www.fmri.org/schema/airsn.xsd"
          xmlns="http://www.fmri.org/schema/airsn.xsd"
          xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:simpleType name="Image">
    <xs:restriction base="xs:string"/>
  </xs:simpleType>
  <xs:simpleType name="Header">
    <xs:restriction base="xs:string"/>
  </xs:simpleType>
  <xs:complexType name="Volume">
    <xs:sequence>
      <xs:element name="img" type="Image"/>
      <xs:element name="hdr" type="Header"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="Run">
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:element name="v" type="Volume"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```
(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" );
}

(Run or) reorientRun (Run ir, string direction) {
    foreach Volume iv, i in ir.v {
        or.v[i] = reorient(iv, direction);
    }
}
**Expressiveness**  
Lines of code with different encodings

<table>
<thead>
<tr>
<th>AppIn</th>
<th>Script</th>
<th>Generator</th>
<th>Swift Script</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATLAS1</td>
<td>49</td>
<td>72</td>
<td>6</td>
</tr>
<tr>
<td>ATLAS2</td>
<td>97</td>
<td>135</td>
<td>10</td>
</tr>
<tr>
<td>FILM1</td>
<td>63</td>
<td>134</td>
<td>17</td>
</tr>
<tr>
<td>FEAT</td>
<td>84</td>
<td>191</td>
<td>13</td>
</tr>
<tr>
<td>AIRSN</td>
<td>215</td>
<td>~400</td>
<td>34</td>
</tr>
</tbody>
</table>
### Expressiveness

**Lines of code with different encodings**

<table>
<thead>
<tr>
<th>AppIn</th>
<th>Script</th>
<th>Generator</th>
<th>Swift Script</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATLAS1</td>
<td>49</td>
<td>72</td>
<td>6</td>
</tr>
<tr>
<td>ATLAS2</td>
<td>97</td>
<td>135</td>
<td>10</td>
</tr>
<tr>
<td>FILM1</td>
<td>63</td>
<td>134</td>
<td>17</td>
</tr>
<tr>
<td>FEAT</td>
<td>84</td>
<td>191</td>
<td>13</td>
</tr>
<tr>
<td>AIRSN</td>
<td>215</td>
<td>~400</td>
<td>34</td>
</tr>
</tbody>
</table>

Collaboration with James Dobson, Dartmouth [SIGMOD05]
Dynamic Provisioning: Swift Architecture

**Specification**
- Abstract computation
- SwiftScript Compiler
- Virtual Data Catalog

**Scheduling**
- Execution Engine (Karajan w/ Swift Runtime)
- C C C C Swift runtime callouts
- Status reporting

**Execution**
- Virtual Node(s)
- launcher App F1
  - Provenance data
  - file1
- launcher App F2
  - Provenance data
  - file2
- launcher App F3
  - Provenance data
  - file3

**Provisioning**
- Falkon Resource Provisioner
- Amazon EC2

Yong Zhao, Mihael Hatigan, Ioan Raicu, Mike Wilde, Ben Clifford
Swift Runtime System

- **Runtime system for SwiftScript** [SSDBM02,CIDR03,Springer06]
  - Populate, update, query virtual data products
  - Schedule, monitor, execute resulting computation on distributed Grid resources
  - Annotate virtual data products with customized metadata
  - Trace provenance of virtual data products

- **Grid scheduling and optimization**
  - Lightweight execution engine: Karajan
  - Dynamic resource provisioning
  - Site selection, data movement, caching
  - Pipelining, clustering, load balancing
  - Fault tolerance, exception handling

A Virtual Data System for Representing, Querying & Automating Data Derivation [SSDBM02]
Swift uses Karajan Workflow Engine

- Fast, scalable threading model
- Suitable constructs for control flow
- Flexible task dependency model
  - “Futures” enable pipelining
- Flexible provider model allows for use of different run time environments
  - Job execution and data transfer
  - Flow controlled to avoid resource overload
- Workflow client runs from a Java container

Synthetic Benchmark

- 18 Stages
- 1000 tasks
- 17820 CPU seconds
- 1260 total time on 32 machines

Ioan Raicu & Yong Zhao, U.Chicago
Release after 180 Seconds Idle
Montage

Yong Zhao and Ioan Raicu, U.Chicago
Molecular Dynamics
Application Example: ACTIVAL: Neural Activation Validation

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, U. Hasson, UChicago.
SwiftScript Program ACTIVAL – Datatypes & Utilities

type script {}  

type brainMeasurements{}  

type precomputedPermutations{}  

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 likelihoods
(brainClusterTable t) bricCentralize ( brainClusterTable bc[]) {
    app { bricCentralize @filenames(bc); }
}
**ACTIVAL: 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);
   }
}
// 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 application

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

clusterThresholdsTable = bricCentralize (randCluster.c);

brainResult = makebrain(origBrain,clusterThresholdsTable,brainFile,specFile);
Example Performance Optimizations
Example Performance Optimizations

Pipelining
Example Performance Optimizations

Pipelining + clustering
Example Performance Optimizations

Pipelining + provisioning
<table>
<thead>
<tr>
<th>Application</th>
<th>#Jobs/computation</th>
<th>Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATLAS*</td>
<td>500K</td>
<td>1</td>
</tr>
<tr>
<td>HEP Event Simulation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fMRI DBIC*</td>
<td>100s</td>
<td>12</td>
</tr>
<tr>
<td>AIRSN Image Processing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FOAM</td>
<td>2000 (core app runs 250 8-CPU jobs)</td>
<td>3</td>
</tr>
<tr>
<td>Ocean/Atmosphere Model</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GADU*</td>
<td>40K</td>
<td>4</td>
</tr>
<tr>
<td>Genomics: (14 million seq. analyzed)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HNL</td>
<td>500</td>
<td>4</td>
</tr>
<tr>
<td>fMRI Aphasia Study</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NVO/NASA*</td>
<td>1000s</td>
<td>16</td>
</tr>
<tr>
<td>Photorealistic Montage/Morphology</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QuarkNet/I2U2*</td>
<td>10s</td>
<td>3-6</td>
</tr>
<tr>
<td>Physics Science Education</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RadCAD*</td>
<td>1000s</td>
<td>5</td>
</tr>
<tr>
<td>Radiology Classifier Training</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIDGrid</td>
<td>100s</td>
<td>20</td>
</tr>
<tr>
<td>EEG Wavelet Proc, Gaze Analysis, ...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SDSS*</td>
<td>40K, 500K</td>
<td>2, 8</td>
</tr>
<tr>
<td>Coadd, Cluster Search</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Fast Ocean Atmosphere Model

160 ensemble members = 2.5 months to run

NCAR
Manual config, execution, bookkeeping

250 ensemble members = 4 days to run

VDS on Teragrid
Automated

Visualization courtesy Pat Behling and Yun Liu, UW Madison
The Swift System
www.ci.uchicago.edu/swift

- 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
  - Lightweight threading, dynamic provisioning, Grid interfaces, pipelining, load balancing

+ Rigorous provenance tracking and query
  - Virtual data schema & automated recording

→ Improved usability and productivity
  - Demonstrated in numerous applications