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: 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
  \[\rightarrow\] 90,000 volumes, 60 GB raw
  \[\rightarrow\] 1.2 million files processed
- 100s of such studies in total

www.fmridc.org
Many Users Analyze fMRI Data

- Wide range of analyses
  - Testing, interactive analysis, production runs
  - Data mining
  - Parameter studies
Three Obstacles to Creating a Community Resource

- **Accessing messy data**
  - Idiosyncratic layouts & formats
  - Data integration a prerequisite to analysis

- **Implementing complex computations**
  - Expression, discovery, reuse of analyses
  - Scaling to large data, complex analyses

- **Making analysis a community process**
  - Collaboration on both data & programs
  - Provenance: tracking, query, application
The **Swift** Solution
(Or: Outline of this Talk)

- **Accessing messy data**
  - Idiosyncratic layouts & formats
  - Data integration a prerequisite to analysis

- **Implementing complex computations**
  - Expression, discovery, reuse of analyses
  - Scaling to large data, complex analyses

- **Making analysis a community process**
  - Collaboration on both data & programs
  - Provenance: tracking, query, application
The **Swift** Solution
(Or: Outline of this Talk)

- **Accessing messy data**
  - Idiosyncratic layouts & formats
  - Data integration a prerequisite to analysis

- **Implementing complex computations**
  - Expression, discovery, reuse of analyses
  - Scaling to large data, complex analyses

- **Making analysis a community process**
  - Collaboration on both data & programs
  - Provenance: tracking, query, application
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 (2)

- Heterogeneous storage format & access protocols
  - Same dataset can be stored in text file, spreadsheet, database, ...
  - Access via filesystem, DBMS, HTTP, WebDAV, ...
- Metadata encoded in directory and file names
- Hinders program development, composition, execution
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 & semantics

- **Logical object**
  - ADO [Microsoft01], in-memory relational model
  - SDO [Beatty,Brodsky03], logical data model for J2EE
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, …
The **Swift** Solution
(Or: Outline of this Talk)

- **Accessing messy data**
  - Idiosyncratic layouts & formats
  - Data integration a prerequisite to analysis

- **Implementing complex computations**
  - Expression, discovery, reuse of analyses
  - Scaling to large data, complex analyses

- **Making analysis a community process**
  - Collaboration on both data & programs
  - Provenance: tracking, query, application
SwiftScript

- Typed parallel programming notation
  - 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

```swift
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;
}
```

Simplified version of fMRI AIRSN Program (Spatial Normalization)
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>

Collaboration with James Dobson, Dartmouth [SIGMOD05]
## Expressiveness

### Lines of code with different encodings

<table>
<thead>
<tr>
<th>Appln</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]
The **Swift** Solution
(Or: Outline of this Talk)

- **Accessing messy data**
  - Idiosyncratic layouts & formats
  - Data integration a prerequisite to analysis

- **Implementing complex computations**
  - Expression, discovery, reuse of analyses
  - Scaling to large data, complex analyses

- **Making analysis a community process**
  - Collaboration on both data & programs
  - Provenance: tracking, query, application
Swift Runtime System

- Runtime system for SwiftScript
  - Translate programs into task graphs
  - Schedule, monitor, execute task graphs on local clusters and/or distributed Grid resources
  - Annotate data products with provenance metadata
- Grid scheduling and optimization
  - Lightweight execution engine: **Karajan**
  - **Falkon**: lightweight dispatch, dynamic provisioning
  - Grid execution: 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 Runtime: Related Work

- Multi-level scheduling [Banga99, Stankovic99]
- Condor glidein [Frey02], Condor Brick [Singh05, Mehta06], MyCluster [Walker06]
- Adaptive resource control [Appleby01], [Ramakrishnan06]
- Lightweight dispatch [Anderson04]
Swift Architecture

Specification
- Abstract computation
- SwiftScript Compiler
- Virtual Data Catalog

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

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

Provisioning
- Falkon Resource Provisioner
- Amazon EC2
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
Karajan Futures
Enable Pipelining

(Dispatch is performed here via GRAM+PBS)
Karajan Futures
Enable Pipelining

(Dispatch is performed here via GRAM+PBS)
Swift Can Use **Falkon** Dispatcher & Provisioner

- Falkon provisioner:
  - Monitors **demand** (incoming user requests)
  - Manages **supply**: selects resources; creates executors (via Globus GRAM)
  - Various decision strategies for acquisition and release

- Falkon executor:
  - Streamlined task dispatch
  - Driven by Karajhan

- Dispatch to other executors also supported—e.g., GRAM

Ioan Raicu, U.Chicago
Falkon Dispatch: Throughput, Scalability

Ioan Raicu, U.Chicago
Falkon Provisioning: Synthetic Benchmark

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

Ioan Raicu & Yong Zhao, U.Chicago
Release after 15 Seconds Idle

- Allocated
- Registered
- Active

Time (sec)

# of Workers

1735.62

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

![Graph showing the number of workers over time with different states: Allocated, Registered, and Active. The graph highlights the time after which the system releases workers.](image)

Ioan Raicu & Yong Zhao, U.Chicago
Swift Application Performance: fMRI Task Graph

Yong Zhao and Ioan Raicu, U.Chicago
Swift Application Performance: fMRI Task Graph

Yong Zhao and Ioan Raicu, U.Chicago
Swift Application

B. Berriman, J. Good (Caltech)
J. Jacob, D. Katz (JPL)
Montage

Yong Zhao and Ioan Raicu, U.Chicago
Other Swift Applications
Include ... 

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

* Using predecessor **Virtual Data System** (VDS)

+ Collaborative science learning & education: 18 experiments, 51 universities/labs, 500+ schools, 100,000 students
The **Swift** Solution
(Or: Outline of this Talk)

- Accessing messy data
  - Idiosyncratic layouts & formats
  - Data integration a prerequisite to analysis
- Implementing complex computations
  - Expression, discovery, reuse of analyses
  - Scaling to large data, complex analyses
- Making analysis a community process
  - Collaboration on both data & programs
  - Provenance: tracking, query, application
Virtual Data Concept

- **Capture information about relationships among**
  - Data (varying locations and representations)
  - Programs (& inputs, outputs, constraints)
  - Computations (& execution environments)

- **Apply this information to:**
  - Discovery of data and programs
  - Computation management
  - Provenance
  - Planning and scheduling
  - Performance optimization

A Virtual Data System for Representing, Querying & Automating Data Derivation [SSDBM02]
Provenance – Related Work

- **Database**
  - Determine the source of tuples [Cui, Widom00]
  - Why and where [Buneman, Khanna01]

- **Scientific**
  - Logbook [Myers, Chappell03] [Bourilkov, Khandelwal06]

- **Service**
  - P-assertions [Szomszor, Moreau03]

- **Other**
  - PASS [Muniswamy-Reddy, Holland06]
Provenance Model

- **Temporal aspect**
  - **Prospective** provenance
    - Recipes for how to produce data
    - Metadata annotations about procedures and data
  - **Retrospective** provenance \[\text{[GCE06]}\]
    - Invocation records of run time environments and resources used: site, host, executable, execution time, file stats ...

- **Dimensional aspect**
  - **Virtual data** relationships
  - **Derivation** lineage
  - **Metadata** annotations

Applying the Virtual Data Provenance Model \[\text{[IPAW06]}\]
Virtual Data Schema
Query Context: fMRI Analysis

First Provenance Challenge, http://twiki.ipaw.info/ [CCPE06]
Query Examples

- **Query by procedure signature**
  - Show procedures that have inputs of type `subjectImage` and output types of `warp`

- **Query by actual arguments**
  - Show `align_warp` calls (including all arguments), with argument `model=rigid`

- **Query by annotation**
  - List anonymized subject images for young subjects:
    - Find datasets of type `subjectImage`, annotated with `privacy=anonymized` and `subjectType=young`

- **Basic lineage graph queries**
  - Find all datasets derived from dataset ‘5a’

- **Graph pattern matching**
  - Show me all output datasets of `softmean` calls that were aligned with `model=affine`

- **Multi-dimensional query**
Acknowledgements

- Swift effort is supported by NSF (I2U2, iVDGL), NIH, UChicago/Argonne Computation Institute
- Swift team
  - Ben Clifford, Ian Foster, Mihael Hategan, Veronika Nefedova, Ioan Raicu, Mike Wilde, Yong Zhao
- Java CoG Kit
  - Mihael Hategan, Gregor Von Laszewski, and many collaborators
- User contributed workflows and application use
  - I2U2, ASCI Flash, U.Chicago Molecular Dynamics, U.Chicago Radiology, Human Neuroscience Lab
Future Work

- **XDTM**
  - Support for services as well as applications
  - Greater abstraction in mappers; databases

- **SwiftScript**
  - Exceptions
  - Event-driven dispatch & execution

- **Falkon**
  - Scale to more resources; data caching
  - Support for service workloads

- **VDC**
  - Integration into Swift; collaboration support
  - Experiments at scale
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
  - Virtual data schema & automated recording

→ Improved usability and productivity
- Demonstrated in numerous applications

http://www.ci.uchicago.edu/swift
Thank You!
Extra Slides
# XDTM – Related Work

<table>
<thead>
<tr>
<th>Features\Model</th>
<th>XDTM</th>
<th>DFDL</th>
<th>PADS/PADX</th>
<th>SDO</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Format</strong></td>
<td>Any</td>
<td>Binary file format</td>
<td>Ad hoc data source</td>
<td>Any</td>
</tr>
<tr>
<td><strong>Abstract Data Model</strong></td>
<td>Declarative description XML Schema</td>
<td>XML Schema</td>
<td>Declarative description XML Schema</td>
<td>Disconnected data graph</td>
</tr>
<tr>
<td><strong>Physical Information</strong></td>
<td>Mapping descriptor</td>
<td>Annotation embedded in XML schema</td>
<td>Mixed with declaration</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Logical/Physical Separation</strong></td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Mapping Specification</strong></td>
<td>Mapping interface</td>
<td>No</td>
<td>No</td>
<td>Data Access Service</td>
</tr>
</tbody>
</table>
VDS – System Diagram

VDL Program

Planner

Execution Plan

Workflow Enactor

Provider

Virtual Data Catalog

Abstract Workflow

Provenance Collector

Workflow Generator

Grid

Launcher
Application – CMS Event Analysis

A e-workspace of simulated data is created for future use by scientists...

Collaboration with Rick Cavanaugh, Dimitri Bourilkov et al. University of Florida [CHEP03]
ATLAS Large Scale Simulation

“How much compute time was delivered?”
(BNL, 1475+ CPUs)

<table>
<thead>
<tr>
<th>CPU years</th>
<th>Jobs</th>
<th>Month</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.45</td>
<td>1402</td>
<td>2004-06</td>
</tr>
<tr>
<td>19.88</td>
<td>13267</td>
<td>2004-07</td>
</tr>
<tr>
<td>33.88</td>
<td>20678</td>
<td>2004-08</td>
</tr>
<tr>
<td>40.06</td>
<td>20229</td>
<td>2004-09</td>
</tr>
<tr>
<td>15.21</td>
<td>30833</td>
<td>2004-10</td>
</tr>
<tr>
<td>14.95</td>
<td>34591</td>
<td>2004-11</td>
</tr>
</tbody>
</table>
Fast Ocean Atmosphere Model

NCAR
Manual config, execution, bookkeeping

VDS on Teragrid
Automated

Visualization courtesy Pat Behling and Yun Liu, UW Madison
Radiology Example

type Image {}

type Center {}

type ROI { Image image; Center center; }

type ROIVec {
    ROI roi[];
}

(AzInfo az) LDAClassify (ROIVec malROIs, ROIVec benROIs, Parameter param, FeatureNames fn) {
    ....
}

ROIVec malROIs<roi_mapper;location="malROI/">;
ROIVec benROIs<roi_mapper;location="benROI/">;
Parameter param<"SegNEExtract.params">;
FeatureNames featureList<"feat-names.lst">;

AzInfo az<"LDA_Az.out">;
az = LDAClassify(malROIs, benROIs, param, featureList);
An International Community

Dataset Requests by Country

Dataset Requests by Cognitive Domain

Using Swift

SwiftScript

Workflow Status and logs

site list

app list

Data

Provenance data

Worker Nodes

launch App a1

f2

launch App a2

f3
Virtual Data Concept (1)

- Capture information about relationships among:
  - Data (varying locations and representations)
  - Programs (& inputs, outputs, constraints)
  - Computations (& execution environments)

- Apply this information to:
  - Discovery of data and programs
  - Computation management
  - Provenance
  - Planning and scheduling
  - Performance optimization

A Virtual Data System for Representing, Querying & Automating Data Derivation [SSDBM02]
Virtual Data Concept (2)

- **Location transparency**
  - Data processing independent of location
  - Replica location service, selection service
- **Materialization transparency**
  - Recipes for data derivation
- **Physical representation transparency**
  - Logical descriptions and relations
Virtual Data System (VDS)

- Introduced Virtual Data Language (VDL)
  - A location-independent parallel language
- Several planners, e.g.:
  - Pegasus: main production planner
  - Euryale: experimental “just in time” planner
  - GADU/GNARE: user application planner (D. Sulahke, Argonne)
- Provenance
  - Kickstart: app launcher and tracker
  - VDC: virtual data catalog

A Virtual Data System for Representing, Querying & Automating Data Derivation [SSDBM02]
VDL/VDS Limitations

- **Missing language features**
  - Data typing & data mapping
  - Iterators & control-flow constructs

- **Run time complexity in VDS**
  - State explosion for data-parallel applications
  - Computation status hard to provide
  - Debugging information complex & distributed

- **Performance**
  - Still many runtime bottlenecks
Swift 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 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 likelihoods
(brainClusterTable t) bricCentralize (brainClusterTable bc[]) {
  app { bricCentralize @filenames(bc); }
}
// 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: Main 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 application

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

clusterThresholdsTable = bricCentralize (randCluster.c);

brainResult = makebrain(origBrain,clusterThresholdsTable,brainFile,specFile);
Sloan Digital Sky Survey

<table>
<thead>
<tr>
<th>Sky region processed</th>
<th>450 sq deg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Num of sky fields</td>
<td>14,000</td>
</tr>
<tr>
<td>Num of procedure calls</td>
<td>5,000</td>
</tr>
<tr>
<td>Num of files</td>
<td>120,000</td>
</tr>
<tr>
<td>Num of Grid sites</td>
<td>4</td>
</tr>
<tr>
<td>Num of Processors used</td>
<td>500 total</td>
</tr>
<tr>
<td>Galaxy cluster identified</td>
<td>60,000</td>
</tr>
</tbody>
</table>

5 days vs. 10 months for the whole dataset (7000 sq deg)
Web Interface

What can you learn? Choose data and conduct a study.

**Analysis**

Performance Study - Look at data from a detector. Can you trust the data?

Lifetime Study - Do you live in Newton's or Einstein's world?

Flux Study - The rain of particles has many interesting properties including its flux. Are there more in Colorado than there are in South Carolina?

Shower Study - You can detect an air shower using the four panels at your school. Your colleagues at other schools will want to know when you detect one, so they can check for coincident showers at their school. Contribute to cutting-edge research on the origin of high-energy primary cosmic rays.

**Management**

**VIEW**

Data Files - See what data has been uploaded into the system.

Plots - Look at what you and other groups have found.

Posters - View and create posters of your plots.

**DELETE**

Data Files - Delete data your group has uploaded.

Plots - Delete plots your group owns.

Posters - Delete posters your group has made.

**Grids** - These investigations are brought to you by grid computing.

Collaboration with Marge Bardeen, Tom Jordan, Liz Quigg, Eric Gilbert, Paul Nepywoda, Fermilab [CCGRID05] [FGCS05]
Sample Program
Web Services

- WS described/imported as procedures
- Dynamic invocation
- XSLT as glue