Data management and data processing using UNICORE

Bernd Schuller
Jülich Supercomputing Centre (JSC)
b.schuller@fz-juelich.de
September 24, 2015
ScaDS Herbstschule, Dresden
Outline

- UNICORE
  - Overview, services, security, clients, ...
- Data management
  - Services, storage backends, ...
  - Demo
- Processing
  - Jobs, workflows and all that
  - Demo
  - Data-driven processing
  - Demo
Local batch system LoadLeveller

- Login/Password
- qsub, qstat, mpirun, ...
- /usr/local/apps/myapp/bin/myapp, ...
- ~/mydata/2011/job123/ergebnisse.txt, ...

ssh / scp
How can I ...

- use multiple, heterogeneous systems seamlessly,
- manage my jobs
- manage input data and results? Metadata?
- across systems? Workflows?
A (subjective) UNICORE timeline

- **1996**: first UNICORE project (Germany only)

- **2002**: **UNICORE 5** → Eurogrid project, UNICORE goes Open Source (I started to work at Jülich on the OpenMolGRID project)

- **2005**: UniGRIDS project: SOAP/WS(RF) interfaces defined,
  **2007**: **UNICORE 6.0** release

- **2014**: **UNICORE 7.0** release
  
  - … we’re still going (thanks to projects and institutional funding)

- **2016**: UNICORE 8 release?
Components at a typical site

- Client
- UNICORE/X server
- Site firewall
- AuthN
- AuthZ
- Front-ends
- Services Logic
- UNICORE/X server
- TSI
A UNICORE Grid ...

- Clients
  - Gateway
    - Service Registry
    - UNICORE/X
  - Gateway
    - Workflow
    - Resource Broker
    - Task Enactor
  - Gateway
    - UNICORE/X
  - Gateway
    - UNICORE/X

UNICORE/X

Slide 8
UNICORE: under the hood
- Workflow enactment
- Task execution
- TargetSystemFactory
- TargetSystem
- JobManagement
- Reservations
- StorageFactory
- StorageManagement
- FileTransfer
- Metadata
- Registry
- Resource Broker
„Factory“ services

1. createTSS()
   TargetSystemFactory creates
   TargetSystemService instance
   UNICORE/X server

2. return TSS address
   use to submit jobs etc

Client

TSI
„Factory“ services: virtualisation support

1. createTSS() provide parameters
2. return TSS address
3. setup VM
4. use when ready, to submit jobs etc
UNICORE Storage Management Service

- File systems
- Apache HDFS
- S3
- iRODS (prototype)
- ...

Client

 mkdir, ls, rm, stat, ...
 upload
download

server-to-server
copy
Storage Management Service

- Initiate file transfers
  - Multi-protocol support

- Metadata management
  - Schema-free, key-value
  - Indexed via Lucene, searchable

- Rule-based data processing
  - New files automatically trigger actions
  - e.g. metadata extraction, compression, etc
StorageFactory service: user-owned storages

- Different types of storage backends can be supported
- User can select and provide required parameters

1. createSMS()
   provide parameters e.g. access keys
2. return SMS address
3. access backend
4. use

StorageFactory

- File system
- HDFS
- S3

StorageManagement service instance
UNICORE Services Environment

- Implemented in Java

  - Very mature and up-to-date services stack
  - SOAP web services
  - REST via JAX-RS

- Numerous other open source libraries
Federated access: security is key
Basic security flow

- User invokes a UNICORE service

  - **Authentication**: who is the user?
    - *Results in the user's X.500 DN ("CN=..., O=..., C="*)
  - **Assign attributes** to the DN
    - *Standard attributes*: role, Unix ID, groups, etc.
    - *Custom attributes*: (e.g. S3 access and secret keys)
  - **Authorisation**
    - *Add context*: e.g. who owns the service?
    - *Check local policies* (XACML)

- **Allow or deny** the request
Delegation

- Allow Service to work on behalf of the user

- UNICORE solution based on SAML
  - Use chain of signed assertions
  - Trust always delegated to particular server
  - Can be validated and audited

1. User submits job
2. Resource A uploads results
   
3. Resource B

Sept 24, 2015 ScaDS Herbstschule, Dresden
End-user authentication in UNICORE

- Pre-UNICORE 7: X.509 client certificates REQUIRED for end-users

- Users tend to hate them
  - All sorts of usage issues

- Lack of understanding leads to lack of security (copying keys to other machines, no/weak encryption, etc)

- Users understand passwords
  - and it is relatively easy to teach basic security measures
Certificate-less end-user authentication

- **No end-user certificates** (not even short-lived)

**Approach**

- Use *signed SAML assertions*
- Issued and signed by the trusted server
- MANY options, e.g. support for existing SAML IdPs, federations like DFN AAI, OAuth2, OpenID Connect, etc
- Flexible solution is required

**Implications**

- Client – server TLS is not client-authenticated any more
- End-user cannot sign anything (no more „non-repudiation“)
Introducing Unity

- Complete **Authentication and Identity Management** solution
- Manage users and user attributes, group membership
- Developed by **ICM / Univ. of Warsaw** (PL)
- Separate product: [www.unity-idm.eu](http://www.unity-idm.eu)
- Already widely deployed: Human Brain Project, EUDAT, ...

![Diagram]

1. credentials
2. authenticate
3. use
Unity architecture

- SAML 2 WS endpoint
- SAML 2 Web endpoint
- Web admin endpoint
- User profile endpoint
- ... ? endpoint
- Local database
- External WS SAML 2 IdP
- External LDAP Service
- External ... ?

Third party IDM systems
Example: authentication assertion

<User>
    <NameID Format="urn:oasis:names:tc:SAML:1.1:nameid-format:X509SubjectName">CN=Demo User,O=UNICORE,C=EU</NameID>
    <SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:sender-vouches">
        <SubjectConfirmationData NotOnOrAfter="2014-11-16T10:30:23.334Z"/>
    </SubjectConfirmation>
</User>

<AttributeStatement>
    <Attribute Name="cn">
        <AttributeValue>Demo User</AttributeValue>
    </Attribute>
    <Attribute Name="email">
        <AttributeValue>test@example.com</AttributeValue>
    </Attribute>
    <Attribute Name="memberOf">
        <AttributeValue>/portal</AttributeValue>
    </Attribute>
</AttributeStatement>

<Assertion>

Name: …
Attributes: …

1. authenticate
1.1. return attributes

X.509

Unity
Resource sharing
Example: user wants to securely share S3 storage

- UNICORE 7.3 introduces per-service ACLs
- Managed by owner of the resource
- Allow read/modify access based on DN, local UID/GID, UNICORE role, VO membership
Single sign-on and resource federation

User

Clients
Portals
APIs

Unity

UNICORE

UNICORE

UNICORE

UNICORE

TSI

UNICORE

hadoop

amazon web services

?
- Batch systems (Torque, Slurm, LoadLeveler, GridEngine, ...)
- Apache Hadoop (YARN)
- Direct execution (e.g. on Windows)

- File systems
- Apache HDFS
- Amazon S3
- ...
- Portals
- Science Gateways
- UCC
- Eclipse-based Rich Client
- RESTful API
- Java APIs
Rich client

- Building, submitting and monitoring jobs and workflows
- Integrated data and storage management
- X.509 and Unity for AuthN
- "Simple view" for novice users
- Based on the Eclipse framework
- Extensibility through plug-ins
- Installation/update mechanism for plug-ins and Application GUIs
UNICORE Portal

- Aim for a simple, easy-to-use web application
- Flexible authentication and user registration
  - support Unity
- Implementation choices
  - Java-based, VAADIN web framework
  - Use UNICORE Java APIs
UNICORE Portal – Job creation view

Job name: Script.job
Select application: Bash shell
Select version: 3.1.16
Command line arguments:

Input parameters
- DEBUG: 
- VERBOSE: 
- OPTIONS: 
- SOURCE: input.sh

Submit
UNICORE Portal – various

- Several „list“ views, e.g. jobs, sites

<table>
<thead>
<tr>
<th>NAME</th>
<th>JOB STATUS</th>
<th>SITE</th>
<th>QUEUE</th>
<th>ESTIMATED FINISH TIME</th>
<th>ACTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job 1</td>
<td>SUCCESSFUL</td>
<td>DEMO-SITE</td>
<td>N/A</td>
<td>unknown</td>
<td></td>
</tr>
<tr>
<td>Test 1</td>
<td>QUEUED</td>
<td>DEMO-SITE</td>
<td>N/A</td>
<td>unknown</td>
<td></td>
</tr>
<tr>
<td>Example job</td>
<td>QUEUED</td>
<td>DEMO-SITE</td>
<td>N/A</td>
<td>unknown</td>
<td></td>
</tr>
</tbody>
</table>

- Workflow creation
- JavaScript
- Initially only simple graphs
UNICORE Portal: Data manager

Select storage

Initiate data movement (direct, not via portal)
RESTful interfaces

REST

- Document / Resource oriented approach
- HTTP semantics (GET, POST, PUT, DELETE)
- Simple JSON (and HTML) data exchange formats and resource representations

- Clients exist in all languages (even `curl` or `wget`)
- Excellent support in Java (JAX-RS)
REST vs SOAP/WS

- **REST**: simpler integration with community solutions / portals
  - e.g. Human Brain Project portal: JavaScript and Python would not play well with SOAP/XML and WS-Security

- **Performance** gains (side effect)
  - SOAP/ XML and WS-Security is rather heavy!

- Stay consistent and backwards compatible
  - Keep SOAP/WS(RF) services
  - Access to same jobs, data, etc
  - Consistent security layer (user DNs, attributes mapping, access control)
Example: job submission

```json
job.u:
{
    Executable: "/bin/echo",
    Arguments: ["Hello World"],
}
```

```
$> curl -X POST -H "Content-Type: application/json"
--data-binary @job.u
https://server:8080/DEMO-SITE/rest/core/jobs
```

```
HTTP/1.1 201 Created
Content-Type: application/json;charset=ISO-8859-1
Date: Mon, 17 Nov 2014 22:08:17 GMT
Location: https://server:8080/DEMO-SITE/rest/core/jobs/ \\n74198236-e970-429d-b55c-a7d59c831f14
```
Example: JSON representation of a job

```json
{
    "owner": "CN=Demo User,O=UNICORE,C=EU",
    "currentTime": "2014-11-17T21:51:49+0000",
    "terminationTime": "2014-12-17T20:09:15+0000",
    "resourceStatus": "READY",
    "status": "SUCCESSFUL",
    "queue": "N/A",
    "submissionTime": "2014-11-17T20:09:15+0000",
    "statusMessage": "",
    "exitCode": "0"

    "_links": {
        "action:abort": {
            "href": "https://localhost:8080/DEMO-SITE/rest/core/jobs/.../actions/abort",
            "description": "Abort"
        },
        "action:restart": {
            "href": "https://localhost:8080/DEMO-SITE/rest/core/jobs/.../actions/restart",
            "description": "Restart"
        }
    }
}
```


---

**Generic part**

**Job specific**

**Links**

---

Sept 24, 2015 ScaDS Herbstschule, Dresden
REST APIs: speedup
some (indicative) performance data

Get resource properties

<table>
<thead>
<tr>
<th>Client threads</th>
<th>Requests/sec WSRF</th>
<th>Requests/sec REST</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>27</td>
<td>79</td>
</tr>
<tr>
<td>2</td>
<td>57</td>
<td>193</td>
</tr>
<tr>
<td>4</td>
<td>80</td>
<td>286</td>
</tr>
<tr>
<td>8</td>
<td>76</td>
<td>332</td>
</tr>
</tbody>
</table>

Job submission

<table>
<thead>
<tr>
<th>Client threads</th>
<th>Requests/sec WSRF</th>
<th>Requests/sec REST</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>34</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>54</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>75</td>
</tr>
</tbody>
</table>
A major use case: the Human Brain Project's HPC platform

1. authenticate
returns token

2. access UNICORE
job submission, data movement, ...

3.1 validate

3. validate token
OK

UNICORE

HBP Molecular Dynamics Supercomputer
HBP Data Analytics Supercomputer
HBP Development System
HBP Supercomputer
HBP Cloud Storage
Data management

- End-users
  - upload/download-sync using various access technologies
- Storages and archives
  - mount filesystems
  - data staging
  - data sharing, data access
- Local (HPC) systems
- Remote compute resources
  - mount filesystems

Sept 24, 2015 ScaDS Herbstschule, Dresden
UNICORE Storage Management Service

- File systems

- Apache HDFS

- S3

- iRODS (prototype)
Storage Management Service

- File system operations

- Initiate file transfers
  - Upload/download, server-to-server
  - Multi-protocol support

- Metadata management
  - Schema-free, key-value, indexed via Lucene, searchable

- Rule-based data-driven processing
  - … more details later
Storages and data movement

- Local Filespace
- Storage instances (HOME, TMP, ...)
- Job directories
- Server-to-server data movement
- Data staging
- Non-UNICORE
- UNICORE Server
File transfer

- Both client-to-server and server-to-server FT available
- Builtin: HTTPS based transfer („BFT“)
  - Single open port needed, (almost) full UNICORE security
  - Simple interface (bulk write, read supports byte ranges), 
    **decent performance** (several MB/sec.)
- Builtin: OGSA ByteIO (uses SOAP messages)
  - Single port, full UNICORE security
  - Rich interface (POSIX-like, block read/write, etc), slow 
    (~400kB/sec)
- **High-performance** UFTPD file server available as an extension
Additional options for data staging

- GridFTP
  - Uses existing globus-url-copy
  - Proxy generated on the client and sent with the job
- Plain HTTP and HTTPS
- FTP and SCP (including client credentials)
- „mailto“ for stage-out :-(
Integrated storage management in the UNICORE Rich Client

- Create files
- Drag and drop from/to desktop environment
- Copy and paste
- Remote file editing
Metadata management

- De-centralized approach: „metadata management service“ (MMS) associated with each storage service („SMS“)
- Schema-free: metadata is key-value pairs
- User can create, edit, delete metadata, trigger automated extraction
- Metadata indexing
Metadata management: implementation

- Metadata storage directly as files on the storage
- Uses well-known open source libraries
- Indexer and search engine: Apache Lucene
- Metadata extraction framework: Apache Tika
- Supported via UCC, URC, REST APIs
  - Example: list file properties including metadata
Integrated metadata management in the UNICORE Rich Client

- Edit
- Trigger auto-extraction
- Search
Demo: storages and metadata using URC
Data processing

- External users, distributed computing, federations
- Remote compute resources
- Local (HPC) systems
- Data staging
- Mounted filesystems
- End-users
- Launch and manage jobs
- Storages and archives
Data processing using UNICORE

- Single jobs
- Workflow system
- Data-driven
- End-user clients
  - URC / UCC / Portal
  - REST APIs
  - Java APIs
Single jobs

- Batch job oriented
  - Data stage-in
  - Execution
  - Data stage-out

- End-user must ...
  - Setup job definition
  - Select site (broker is available)
  - Upload input data from local machine
  - Submit
Single job execution

1. submit
1.2 return job address
1.1 create

3. start

4. wait until done

2. import data
5a. export data
5b. stage-out data

2. stage-in data

Target System Service

Client

Local Filespace

Job

USpace

Remote Storage Spaces
Workflow system

- Sequences / Graphs / Control
- Based on single jobs
- End-user client tasks
  - Setup workflow definition
  - Upload input data
  - Submit

- Pros
  - Easy automation of complex processes
  - Control constructs available
  - Low load on client side
  - Powerful GUI client

- Cons
  - High overhead on servers
  - Data staging can be a limiting factor
Demo: jobs and workflows using URC
„Data-driven“ processing

- As opposed to „job-oriented“
- Driven purely by data
- No end-user involvement required (apart from setup)
- Kind of like a „cron job“
Basic UNICORE architecture - I

- User centric
- Everything is „owned“ by a user (submission services, jobs, storages, file transfers ...)
- Fully compatible with Unix file permissions
- UNICORE never operates as a „superuser“

Client

invokes services
provides credentials
delegates trust

Services
(UNICORE(X))
Basic UNICORE architecture - II

- Services / logic lives on the UNICORE/X server
- File system and batch system accessed via TSI agent
- TSI accessed via request/response
- No file system notifications possible with current TSI
UNICORE Storage Management Service

- File systems
  - Apache HDFS
  - S3
  - iRODS (prototype)

Client

mkdir, ls, rm, stat, ...

upload download

server-to-server copy
UNICORE SMS
Storage instance(s)

/.UNICORE_Rules
/directory1
/directory2
/...

1. Trigger event
Evaluate rules

2. Create actions

3. Run / submit
Actions

4. Produce new data

Scripts
Batch jobs
...
Periodic directory scan

- Files can be written independently of UNICORE
- Scan interval configurable
- Directory include/exclude patterns
Types of actions

- Local script
  - Executed via TSI
  - TSI node (cluster login node)

- Local batch job
  - Executed via XNJS/TSI
  - Compute node(s)
  - UCC-like job description

- Metadata extraction
Required setup

- Create a storage (service instance) where data-driven processing is enabled
  - by the user
  - pre-configured by the admin

- Configure (edit .UNICORE_Rules file)
Example

- Goal: calculate checksums (md5) of PDF files in a certain directory using batch jobs

- Rule (job is defined in the usual JSON job syntax)

  Name: computeMD5Sum, Match: ".*\.pdf",
  Action: {
    Type: BATCH,
    Job: {
      Executable: "/usr/bin/md5sum",
      Arguments: ["${UC_FILE_PATH}"],
      Exports: [
        {From: "stdout",
        To: "file://${UC_BASE_DIR}/checksums/${UC_FILE_NAME}.md5"},
      ],
    }
  }
Summary
A federation software suite

- Secure and seamless access to compute and data resources
- Excellent application and workflow support
- Wide variety of clients: GUI, commandline, APIs, ...
- Portable code, supports UNIX, MacOS, Windows and many resource management systems (Torque, Slurm, SGE, …)
- Easy to install, configure, administrate and monitor
- Active developer team, responsive to user wishes ;-), quick and efficient support
- Open source, BSD licensed, visit http://www.unicore.eu
Team / Thank you

- Björn Hagemeier, Valentina Huber, André Giesler, Boris Orth, Maria Petrova, Jedrzej Rybicki, Rajveer Saini and many others at JSC
- Krzysztof Benedyczak, Marcelina Borcz, Rafał Kluszczynski, Piotr Bała and others at ICM / Warsaw University
- Richard Grunzke and others at Technical University Dresden
- Students: Burak Bengi, Maciej Golik, Konstantine Muradov
- … many others who reported bugs, suggested features, contributed code and provided patches