Secure High-Throughput Computing Using UNICORE XML Spaces

UNICORE Summit 2010

Zellescher Weg 12
Willers-Bau A306
Tel. +49 351 - 463 - 37903

Richard Grunzke (Richard.Grunzke@tu-dresden.de)
Bernd Schuller (B.Schuller@fz-juelich.de)
Contents

- UNICORE
- High-Throughput Computing
- High-Throughput Computing with UNICORE
- Space based approach
- Measurement results
- Summary & Outlook
UNICORE

Computing Grid middleware easing the access to computing resources
Experiments in the area of molecular mechanisms of endocytosis and endosome biogenesis

Parts of the cells are highlighted to examine cellular processes

Automatic microscopes create millions of images of different stages of the processes

Computing power is needed to extract information from images
Analysis exceeds computational capabilities of MPI-CBG

- Analysis at ZIH via command line version of analysis software
- Many short independent single-core jobs running in farming mode
- Running on Deimos (2584 cores) and Emila (512 cores)
- 2009 largest project (30%) on Deimos and 26% in 2010 so far
- Research possibilities limited by computational power
  - ever growing demand
High-Throughput Computing with UNICORE 1/3

1. Creates Job
2. Client Submits Job
3. Forwards Job
4. Runs Job
5. Returns

Workstation

ZIH

UNICORE Middleware

Jülich Supercomputing Centre

others

Emilia

Deimos

Juggle

Richard Grunzke, Bernd Schuller
High-Throughput Computing with UNICORE 2/3

- Submission of 1000 date jobs to test overhead
- „ucc run“
  - One ucc process for every job
  - 2702s for 1000 date jobs, 0.37 jobs/s
  - Not suitable because of program starting overhead
- „ucc batch“
  - One process continuously submits jobs
  - 450s for 1000 date jobs, 2.22 jobs/s
  - About 6 times faster
“ucc batch“ works but problems with
- High CPU utilization on client side
- Efficient use of UNICORE/X is hard
  - Client must decide where to send a job
  - Efficient scheduling needs detailed load information
- Doesn‘t scale with number of resources
- Coupled communication between UCC and UNICORE/X
  - Find appropriate site
  - Submit job
  - Poll for status

More decoupled approach needed
Space Based Approach

- XML document repository ("XML Space") as central pool of jobs
- Client communicates mainly with the Space
- UNICORE/X takes jobs as manageable
XML Space basics

Client

write() -> Tuple Space

read() -> XML Document

take() -> Query template

Richard Grunzke, Bernd Schuller
Space based job processing

```xml
<Job xmlns="...">
  <JobID>my test job</JobID>
  <Status>NEW</Status>
  <JSDL>
    <jsdl:JobDescription>
      <jsdl:Application>...</jsdl:Application>
    </jsdl:JobDescription>
  </JSDL>
</Job>
```

1. write

Client

Tuple Space

2. take

Execution Node

3. process

```
<Job xmlns="...">
  <Status>NEW</Status>
  </Job>
```

4. write

```
<Job xmlns="...">
  <Status>DONE</Status>
  </Job>
```

5. take
Space Based Batch Mode

- „ucc space-batch“ available
- Submitting on client side not CPU-bound anymore
- Significantly less communication between client and middleware
- UNICORE/X optimally adjustable to high throughput and stability
Security 1: Trust delegation

Based on standard SAML trust delegation from client to execution site

Client side

- Client discovers execution nodes from Registry
- Issues delegation and stores it in the Space
- Job contains the client name

Server side

- Server retrieves matching trust delegation, based on server ID and client name
- The jobs are “owned” by the client

Performance considerations

- Client issues delegations valid for a certain period
- Delegations are cached server-side
Security 2: Access control on the Space

- Space entries are „owned“ by the Grid entity, the client, that created them
- Space operations (read, take) need access control beyond the usual UNICORE “per-service” mechanism
  - the Space concept is a collaborative approach, but real life sometimes is not
  - Example: multiple users using Space based job processing
- Implemented ACL check on read(), write(), take()
  - Leverage standard UNICORE mechanisms and concepts
    - XACML callouts
    - Owner, role, …
    - Trust delegation
  - Minimize overhead, check ACL only if entry matches
Security 3: Performance impact of ACL check

- Single ACL callout has low overhead (several milliseconds)
- But: finding the *first accessible space entry* may involve many needless entry lookup operations and ACL checks
- (Future) solution: use entry owner as „primary key“ to only try to match accessible entries
"ucc run" vs "ucc batch" vs "ucc space-batch" - 1000 date jobs
Value of updatePeriod

![Bar chart showing jobs/s vs updatePeriod in ms]
Number of client threads
Measurement results 4/7

ACL checking on vs off

![Bar chart showing ACLs on vs ACLs off](chart.png)
Submitted jobs

*100 = number of jobs
Measurement results 6/7

- 2 Job Taker on 1 UNICORE/X vs
- 4 Job Taker on 1 UNICORE/X vs
- 4 Job Taker on 2 UNICORE/X
Value of bunchsize
Summary

- Highly scalable
- Throughput of 3 jobs/s, very likely to be improved
- Client not bottleneck anymore
- UNICORE/X optimally configurable for stability and efficiency
- Direct file upload from client not possible, need to use stage-in
Outlook

- Further tests to find bottlenecks and limitations
- Integration of multiple space support
- Performance optimizations
Thank you.