BDII Story
An Evolutionary Approach

Laurence Field
CERN

www.eu-egee.org
Contents

• MDS Experience
• BDII Evolution
• Development Tips
  – Fundamentals
  – Key Messages
• **Metadata Directory Service (MDS)**
  - http://www.globus.org/toolkit/docs/2.4/mds/

• **Information Providers (IP)**
  - Scripts that get the information and return LDIF

• **Grid Resource Information Service (GRIS)**
  - Daemon that runs the IP and answers LDAP queries
  - Register to a GIIS

• **Grid Information Index Service (GIIS)**
  - Answers LDAP queries by querying registered GRIS’s or GIIS’s.

• **Both the GRIS and GIIS have a 30s cache**
  - To reduce load and improve performance
• **Query forwarded to lower levels if cache is stale**
  – Complex timings required
    ▪ Which were initial broken
    ▪ Strange behaviour due to timing interaction
  – Lower level problems affected higher levels
    ▪ 1 broken work node could bring down the whole system
      • qstat hung, IP hung, GRIS hung, GIIS hung
  – Response significantly slower if cache is not used
    ▪ Seriously affected query scalability

• **MDS did not work in a distributed environment**
  ▪ Initially would not scale past 4 sites

• **OpenLDAP to the rescue!**
  – Tests showed that a standard LDAP server was more stable
1. Need to decouple query handling and data gathering
2. Use off-the-shelf, established components

• The Berkeley Database Information Index
  – Uses a standard OpenLDAP server as supplied by the OS
    ▪ With the Berkeley database backend
  – Updated by a perl process
    ▪ Configuration file containing LDAP URLs for the sites
    ▪ Use ldapsearch command used as it is stable
      • And should not break with and OS upgrade
  – Balance freshness of information and performance

• BDII first used as top-level GIIS
  – Due to instability problems of the top-level GIIS
    ▪ Is now used at the top, site and resource level
• Initial data gathering done in series
  – Worked with 5 stable sites
  – However, Each “timed-out” site would added 30s.
    ▪ Firewalled ports are very common!
  – Performance testing done using parallel threads.

3. Use parallel thread pool to address timeout failures

LDAP ADD scalability testing
  – ldapadd faster than slapadd
    ▪ drop database and recreate
  – slapadd faster than ldapadd
    ▪ Fastest with bulk updates

3. Use most performant method, not the most elegant
Multiple DBs instances used to increase performance
- Writing to the DB affects the read operations
- Separate read and write
- Read only, write only and one spare for queries to finish
- This functionality is enabled by the port forwarder

List of LDAP sources to query from configuration file
Load Balancing The BDII

DNS Round Robin Alias

Queries
Query Scalability Improvements

- Log queries to production BDIIIs
  - Analyzed log files for production usage
  - Improved client queries
- Designed query performance tests
  - Using production queries as the input
- Evaluated different deployment scenarios
  - Hardware
  - OpenLDAP Versions
  - Slapd configuration
    - Using indexes significantly improved performance

5. Understand and measure the real usage
Performance Improvements

Enabling Grids for E-sciencE

![Graph showing performance improvements](image)

Log Scale!

indexed DB
nonindexed DB

parallel Requests

indexed DB
nonindexed DB

parallel Requests

0,01
0,1
1
10
parallel Requests

EGEE-II INFSO-RI-031688

Laurence.Field@cern.ch

12
Future Work

• Ongoing technology evaluation
  – Is OpenLDAP the most performant technology?

• Improving performance through service deployment
  – Improve load balancing mechanism
    ▪ Alternative handling of state

6. Separate state and connection handling

• Re-evaluate the architecture
  – Can we improve the performance through design?
    ▪ Based on the real usage measured use cases
1. **Scalability**
   - Over subscribed service tend to be brittle

2. **Robustness**
   - \( P_{\text{failure}}(x) \rightarrow 1 \) as \( x \rightarrow \infty \)
Key Messages

• **Keep It Simple Stupid (KISS)**
  – Complexity increases the chance of failure

• **Focus on the core task**
  – Make it robust make it scale
    ▪ Avoid any feature enhancements until the core is robust
  – Understand and measure the production use cases

• **Build upon established technology/standards**
  – Avoid emerging technology/standards
    ▪ and unstable/unproven software
  – Use the appropriate technology
    ▪ XML is not the only solution!
  – We are never the first to solve basic computing problems!

• **Understand the management of connections and state**
  – Well managed state is easy to load balance
    ▪ The different between software and a service