Libarcclient
A powerfull library for interfacing ARC-enabled resources

PUCOWO 2010
Martin Skou Andersen
skou@nbi.dk
University of Copenhagen
Outline

• About me
• The EMI project
• ARC-NOX
• Libarcclient features
• Steps to submit a job
• Submission example
• Java submission example
• Wrap up
About me

• Master in Particle Physics
  – ATLAS experiment
  – Used grid throughout master study

• Developer on the KnowARC project
  – Grid-enabled Know-how Sharing Technology Based on ARC Services and Open Standards (KnowARC)
  – Active in client development

• Developer on the EMI project
  – Team leader of the ARC Compute Client product
European Middleware Initiative (EMI)

• The project (part of FP7) last 3 years with 2336 person months and 24 partners

• It will:
  – Consolidate the middlewares ARC, gLite, UNICORE and dCache
  – Develop and maintain the middleware stack

• The middleware provided by EMI is a candidate for inclusion in UMD provided by EGI
ARC-NOX

• ARC-NOX: New generation ARC middleware
• Meant to replace current ARC
• Not yet production ready
• ARC-NOX was developed in the KnowArc project using C++ programming language
• libarcclient is part of ARC-NOX
  – It is capable of doing job submission and management
  – And are able to do resource discovery
• Should be used in implementations connecting to ARC resources!
ARC-NOX

• ARC-NOX: New generation ARC middleware
• Meant to replace current ARC
• Not yet production ready
• ARC-NOX was developed in the KnowArc project
• libarcclient is part of ARC-NOX
  – It is capable of doing job submission and management
  – And are able to do resource discovery

• Should be used in implementations connecting to ARC resources!
Libarcclient Features

• Supports multiple middlewares
  – Classic ARC
  – Web Service ARC
  – gLite (CREAM)
  – UNICORE
  – Easy extendable through plug-ins
Libarcclient Features

• Supports multiple middlewares
• Understands multiple job description languages
  – xRSL
  – JSDL (Posix, and HPCP)
  – JDL (gLite)
Libarcclient Features

• Supports multiple middlewares
• Understands multiple job description languages
• Supported on major platforms
  – Linux (RedHat, fedora Ubuntu, Debian)
  – Mac OS X
  – Windows
Libarcclient Features

• Supports multiple middlewares
• Understands multiple job description languages
• Supported on major platforms
• Job management
  – Job submission, resubmission and migration
  – Job querying, killing, resuming, renewing, output retrieval, output catenation, ...
Libarcclient Features

• Supports multiple middlewares
• Understands multiple job description languages
• Supported on major platforms
• Job management
• Resource brokering
  – Supports Random, FastestQueue, Data and Benchmark brokers
  – Custom broker can easily be added as plug-in
Libarcclient Features

- Supports multiple middlewares
- Understands multiple job description languages
- Supported on major platforms
- Job management
- Resource brokering
- Language bindings
  - Can be interfaced using Python and Java
  - Bindings generated with SWIG
Libarccclient Features

• Supports multiple middlewares
• Understands multiple job description languages
• Supported on major platforms
• Job management
• Resource brokering
• Language bindings
• Packages which need to be installed
  – nordugrid-arc-nox-client and nordugrid-arc-nox-plugins-base
  – For submission to classic ARC nordugrid-arc-nox-plugins-globus
Libarcclient: Submit Job

- Steps to use libarcclient to submit a job
- Create or parse job description
- Generate or discover computing targets
- Possibly do resource brokering
- Submit job description
Libarcclient: Submit Job

• A job description is represented by the JobDescription class

• class JobDescription
  – It is an extension of JSDL covering xRSL and JDL
  – Can be manipulated directly
  – Or can be populated from a file (xRSL, JSDL or JDL)
  – Can be used for job description translation
  – In future releases it will be possible to easily add a custom job description interpreter
Libarcclient: Submit Job

- Resources can be created and populated by the TargetGenerator class
- class TargetGenerator
  - Initialized by information end-points (URLs)
  - Creates ExecutionTarget instances
- Supports the following targets:
  - Classic ARC computing element (grid-manager)
  - Web Service ARC computing element (A-REX)
  - CREAM service
  - UNICORE service
  - Easy extendable through plug-ins
Libarcclient: Submit Job

• A Resource can be initialised with the ExecutionTarget object

• class ExecutionTarget
  – A flatten structure of a subset of GLUE2
  – Can be manipulated directly
  – Or can be generated by target querying

• Resource brokering is carried out by the Broker class

• class Broker
  – Initialized by list of ExecutionTargets
  – Random, FastestQueue, Data, Benchmark
Submission example

- First initialize UserConfig object
  - Initializes credentials, among others

```cpp
#include <arc/client/ExecutionTarget.h>
#include <arc/client/JobDescription.h>

int main(int argc, char** argv) {
    Arc::UserConfig usercfg;

    Arc::JobDescription jd;
    desc.Parse(argv[1]);

    Arc::ExecutionTarget t;
    t.GridFlavor = "ARC0";
    t.url = argv[2];
    t.Cluster = argv[3];

    Arc::URL jobid = t.GetSubmitter(usercfg)->Submit(jd, t);
    std::cout << "Job submitted with jobid: " << jobid.str() << std::endl;
    return 0;
}
```
Submission example

- Parse job description file (argv[1])

```cpp
#include <arc/client/ExecutionTarget.h>
#include <arc/client/JobDescription.h>

int main(int argc, char** argv) {
    Arc::UserConfig usercfg;

    Arc::JobDescription jd;
    desc.Parse(argv[1]);

    Arc::ExecutionTarget t;
    t.GridFlavor = "ARC0";
    t.url = argv[2];
    t.Cluster = argv[3];

    Arc::URL jobid = t.GetSubmitter(usercfg)->Submit(jd, t);
    std::cout << "Job submitted with jobid: " << jobid.str() << std::endl;
    return 0;
}
```
Submission example

- **Initialize ExecutionTarget**
  - Select Classic ARC for submission
  - Set cluster contact endpoint (argv[2])
  - Set information endpoint (argv[3])

```cpp
Arc::UserConfig usercfg;

Arc::JobDescription jd;
desc.Parse(argv[1]);

Arc::ExecutionTarget t;
t.GridFlavor = "ARC0";
t.url = argv[2];
t.Cluster = argv[3];

Arc::URL jobid = t.GetSubmitter(usercfg)->Submit(jd, t);
std::cout << "Job submitted with jobid: " << jobid.str() << std::endl;
return 0;
```
Submission example

• Get Submitter object and submit job

```cpp
#include <arc/client/ExecutionTarget.h>
#include <arc/client/JobDescription.h>

int main(int argc, char** argv) {
  Arc::UserConfig usercfg;

  Arc::JobDescription jd;
  desc.Parse(argv[1]);

  Arc::ExecutionTarget t;
  t.GridFlavor = "ARC0";
  t.url = argv[2];
  t.Cluster = argv[3];

  Arc::URL jobid = t.GetSubmitter(usercfg)->Submit(jd, t);
  std::cout << "Job submitted with jobid: " << jobid.str() << std::endl;
  return 0;
}
```
Submission example

• Only a simple example
• No sanity checks in example!
• Resource discovery
  – Using resource discovery multiple resources can be made available (index-servers)
• No brokering done
• Standard location of proxy used
import nordugrid.arc.*;

public class arcsub {
   public static void main(String[] args) {
      System.loadLibrary("accARC0");
      System.loadLibrary("jarc");

      UserConfig usercfg = new UserConfig();

      JobDescription job = new JobDescription();
      job.Parse(args[1]);

      ExecutionTarget target = new ExecutionTarget();
      target.GridFlavor = "ARC0";
      target.url = argv[2];
      target.Cluster = argv[3];

      URL jobid = target.GetSubmitter(usercfg).Submit(target, job);
      System.out.println("Job submitted with jobid: " + jobid.str());
   }
}
Wrap up

• Libarcclient easy to use
• Supports different middleware and is easy extensible
• Supported on major platform
• Can be interfaced with C++, Python and Java
• Development and maintainance will be done in the EMI project (3 years)
• Any question about or requests for the libarcclient is very welcome
Thank you

EMI is partially funded by the European Commission under Grant Agreement RI-261611