SwiNG SWISS NATIONAL GRID ASSOCIATION

SGS11: Swiss Grid School 2011

ARC for developers:

Implementing High-Throughput Computing solutions on the Grid

Sergio Maffioletti
Grid Computing Competence Center GC3
University of Zurich
sergio.maffioletti@gc3.uzh.ch

Why development is needed ?

- Most of the current scientific research is dealing with very large number of data to analyze
- This is not just parameter sweep usecase, but in general Hight Throughput Computing (HTC)
- Tools are needed to efficiently address both computing and data handling issues
- Single operation client tools are not always adequate to cope with the high throughput requirements of even a single experiment
- Dedicated end-to-end tools should be developed to tailor end-user requirements (no general solution that fits for all)

Why development is needed ?

- ARC provides API interfaces that could be used to address the computing and data handling part of any end-to-end solution that aim to leverage grid capabilities
- ARC1 provides a revised API model centered around plugins (it provides plugings for ARC0, ARC1 and CreamCE compute elements for example)
- Python bindings turned out to be extremely practical for rapid development of prototype HTC solutions

What is the added value instead of my beloved bash/perl scripts ?

- Using the API allows to directly control and use the data structures ARC provides
  - like a list of computing resources – ExecutionTargets -, or easily create several JobDescription(s) from a template
- It allows to directly manipulate these data structures and/or integrate them in a control driver script
  - for example, it is very easy to obtain a list of Job objects each of them representing a submitted job)
- It allows to have a finer grained control on such data structure
  - like optimized bulk submission minimizing the ldapsearches on remote ends
- It allows to implement own allocation and resubmission strategies

A typical high-throughput use case?

- Run a generic Application on a range of different inputs; where each input is a different file (or a set of files).
- Then collect output files and post-process them, e.g., gather some statistics.
- Typically implemented by a set of sh or Perl scripts to drive execution on a local cluster.

A programming example

From a folder containing 20 .inp input files

Search for the one that has a particular pattern

Each file will be a job submitted

Driver script should handle:

1. All preparatory steps (create one JobDescription per input file)
2. Bulk submission
3. Global control on all submitted jobs
4. Result retrieval
5. Check which job found the pattern

Basic ARC libraries data structures

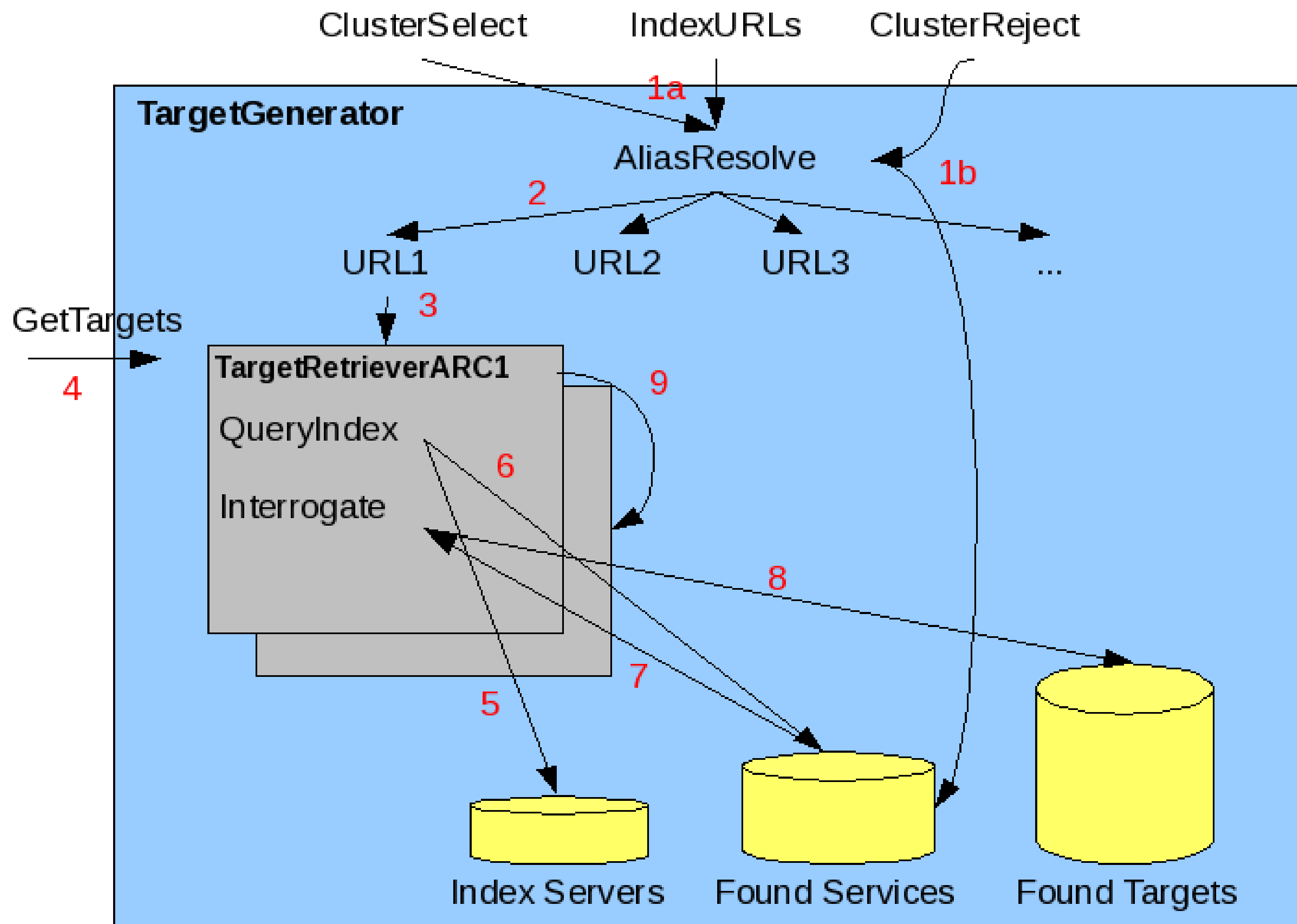
1. User configuration
2. Resource discovery and Information retrieval
3. Job submission
4. Job Management

Resource Discovery and Information Retrieval

- The new libarcclient resource discovery and information retrieval component consists of three classes; the TargetGenerator, the TargetRetriever and the ExecutionTarget.

- TargetRetriever is a base class for further grid middleware specic specialization (plugin)

Resource Discovery and Information Retrieval



SwiNG

TargetGenerator

- The TargetGenerator class is the umbrella class for resource discovery and information retrieval (index servers and execution services).
- The TargetGenerator loads TargetRetriever plugins (which implements the actual information retrieval) from URL objects found in the UserConfig object

```
arc.GetTargetGenerator(_usercfg, 0)
```

SwiNG

RetrieveExecutionTargets(self) GetExecutionTargets(self)

Retrieve available execution services.

The endpoints specified in the UserConfig object passed to this object will be used to retrieve information about execution services (ExecutionTarget objects).

The discovery and information retrieval of targets is carried out in parallel threads to speed up the process. If a endpoint is a index service each execution service registered will be queried.

List of Execution targets can be accessed by invoking

`GetExecutionTargets()`

SwiNG

Job Submission

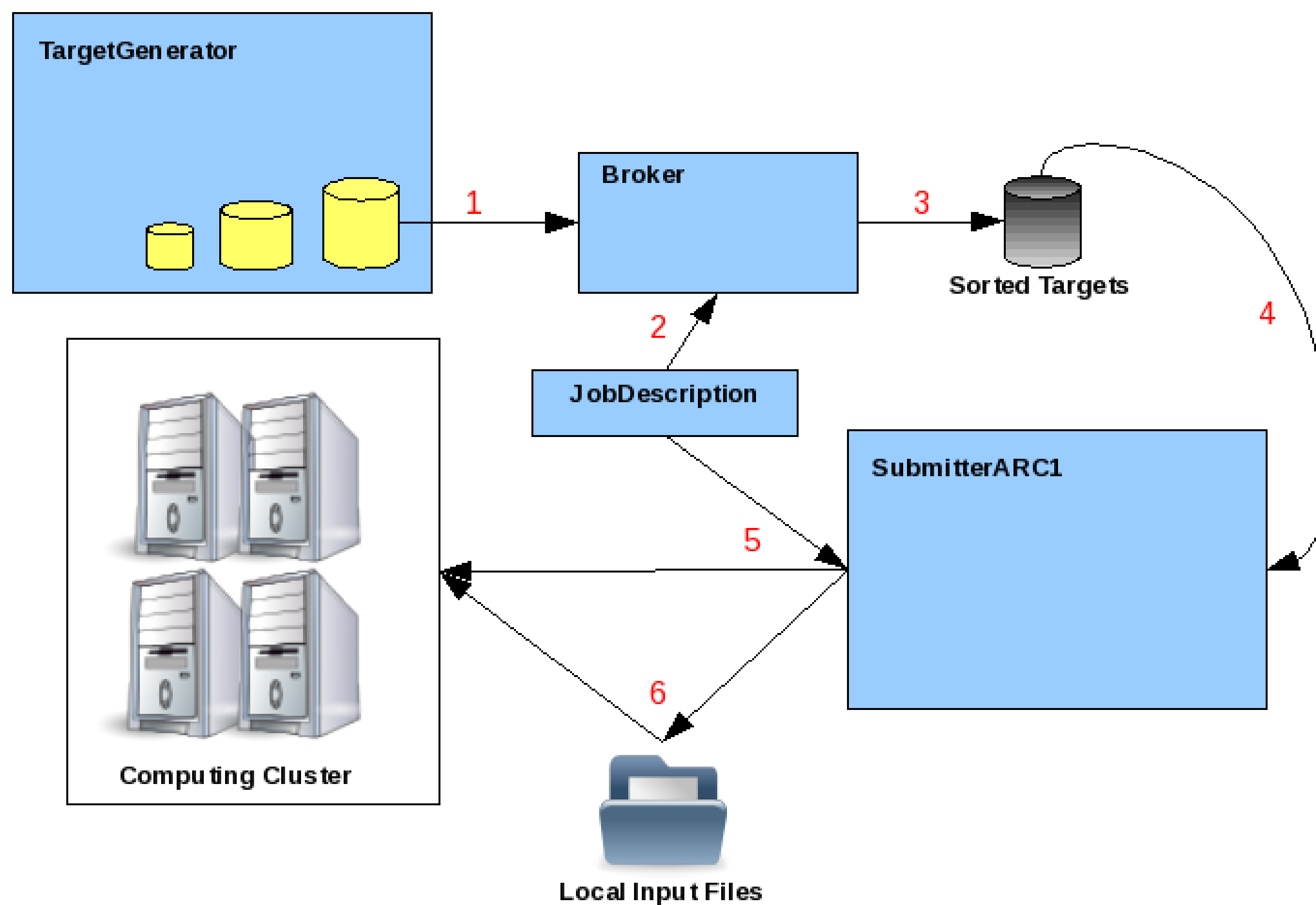
Job submission starts with the resource discovery and target preparation.

Only when a list of possible targets is available the job description is read and brokering method is applied to rank the ExecutionTargets according to the JobDescription's requirements.

Note: this allows to submit bulk of jobs without having to re-perform the resource discovery.

SwiNG

Job Submission



Job Submission

- The TargetGenerator has prepared a list of ExecutionTargets.
- In order to rank the found services (ExecutionTargets) the Broker needs detailed knowledge about the job requirements, thus the JobDescription is passed as input to the brokering process.

```
Broker.PreFilterTargets([ExecutionTargets],
                        JobDescription)
Broker.GetBestTaget()
```

SwiNG

Job Submission

Target.Submit(arc.UserConfig, JobDescription, arc.Job)

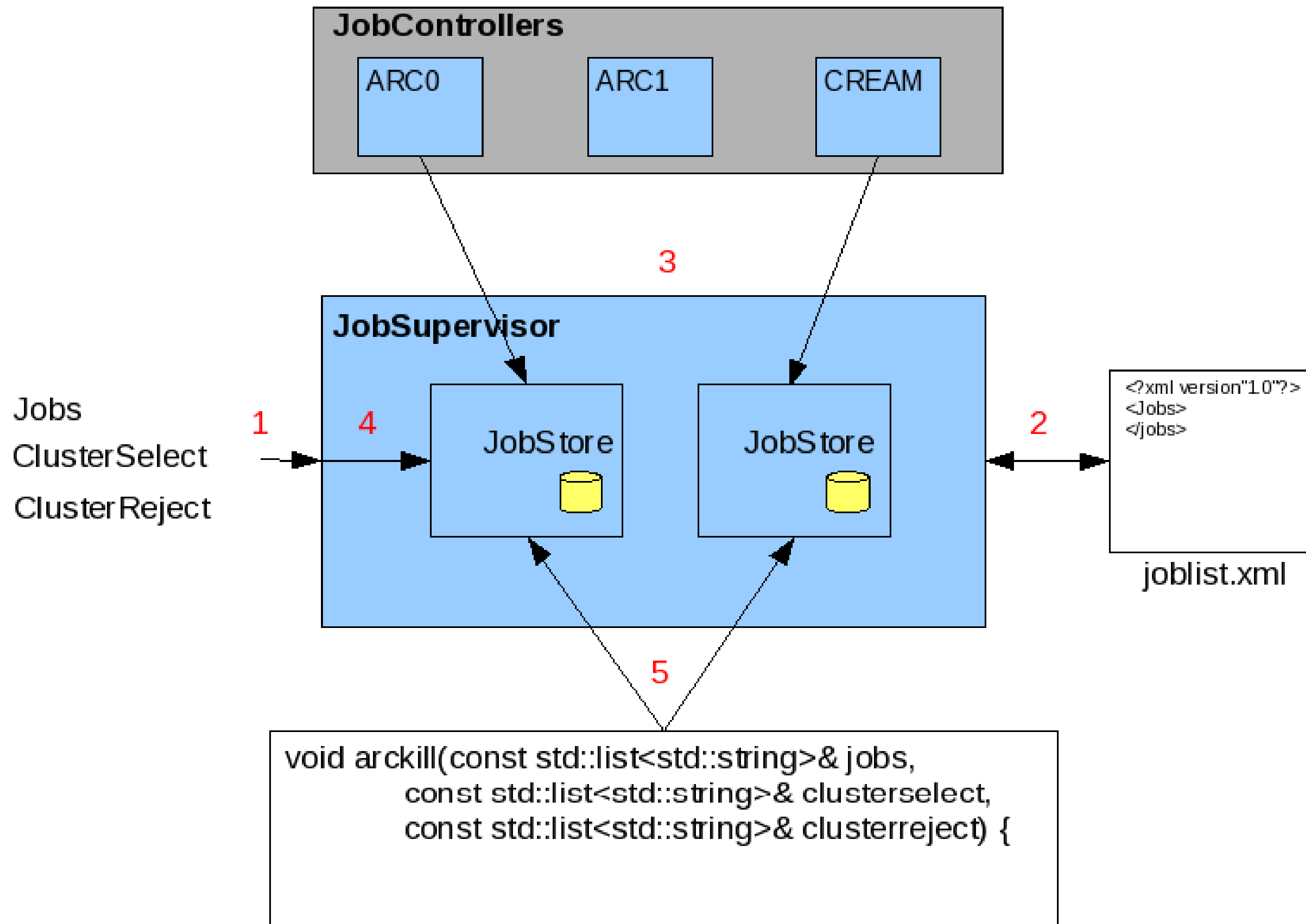
Returns True/False and modifies arc.Job object

Job Management

- Once a job is submitted, job related information (job identication string, cluster etc.) is stored in a local XML (default: $HOME/.arc/jobs.xml).
- This file may contain jobs running on completely different grid flavours, and thus job management should be handled using plugins similar to resource discovery and job submission.
- The job managing plugin is called the JobController and it is supported by the JobSupervisor and Job classes.

SwiNG

Job Management



SwiNG

Job Management

```
jobsupervisor = arc.JobSupervisor(_usercfg, [])
jobsupervisor.GetJobControllers()
```

joblist file is extensively used by the JobSupervisor to identify the JobController flavours which are to be loaded.

Jobs can then be managed through respective JobControllers

SwiNG

sgs2011_arc_htc.py

```
# import User configuration parameters
_usercfg = arc.UserConfig("", "")
_usercfg.ClearSelectedServices()

# fixed values for the purpose of the exercise
arc_version = 'ARC0'
host_endpoint = 'aio.grid.zoo' # ARC_CE hostname

# add computing service
_usercfg.AddServices(["%s:%s" %
```

sgs2011_arc_htc.py

```
_target_generator = arc.TargetGenerator(_usercfg, 0)

# this call spawns remore researches
_target_generator.RetrieveExecutionTargets()

targets = _target_generator.GetExecutionTargets()
```

SwiNG

sgs2011_arc_htc.py

```
jd = arc.JobDescription()
jd.Application.Executable.Name = "/bin/grep"
jd.Application.Executable.arguments = ["-1", "sgs2011",
   "inputfile"]
jd.Application.Output = "sgs2011.out"
jd.Application.Error = "sgs2011.err"
jd.Application.LogDir = ".arc"
jd.Resources.SlotRequirement.NumberOfSlots.max = 1
jd.Resources.IndividualPhysicalMemory.max = 100
jd.Resources.TotalWallTime.range.max = 60
```

SwiNG

sgs2011_arc_htc.py

```
jd_list = []

  # iterate over input folder and create one
jobdescription per file
  for filename in os.listdir(input_folder):
      xrsl = xrsl_template + "(InputFiles=('inputfile'
'%s'))" %
os.path.abspath(os.path.join(input_folder,filename)) +
"(jobname='%s')" % filename

      jd = arc.JobDescription()
      if not arc.JobDescription.Parse(jd, xrsl,
jobdesclang):
          log.error("Failed creating JobDescription
with xrls '%s'" % xrsl)
```

SwiNG

sgs2011_arc_htc.py

```
ld = arc.BrokerLoader()
broker = ld.load("Random", _usercfg)
j = arc.Job()

 for jd in jd_list:
     broker.PreFilterTargets(targets, jd)
     target = broker.GetBestTarget()
     if not target:
         continue # no target found for this
JobDescription
     submitted = target.Submit(_usercfg, jd, j)
     if submitted:
         job_list[j.JobID.str()] = j
```

SwiNG

sgs2011_arc_htc.py

```
    _jobsupervisor = arc.JobSupervisor(_usercfg, [])
    _controllers = _jobsupervisor.GetJobControllers()
...
      for c in _controllers:
           c.GetJobInformation()
           joblist = c.GetJobs()
...
           # Note, joblist is a list of arc.Job object
           for job in joblist:
            # Check and manipulate individual jobs
```

SwiNG

sgs2011_arc_htc.py

```
  download_dir = os.path.join(os.getcwd(),job.Name)
  download_file_list =
   controller.GetDownloadFiles(job.JobID)

  source_url = arc.URL(job.JobID.str())
  destination_url = arc.URL(download_dir)
...
  for remote_file in download_file_list:
...
     controller.ARCCopyFile(source_url,destination_url):
```

Existing projects

- Atlas control tower
- Ganga
- GC3Pie
  - Created at the University of Zurich's Grid Computing Competency Center (GC3) http://www.gc3.uzh.ch/
  - Open-source, hosted at http://gc3pie.googlecode.com gc3pie@googlegroups.com

ARC for sysadmins.
The tutorial

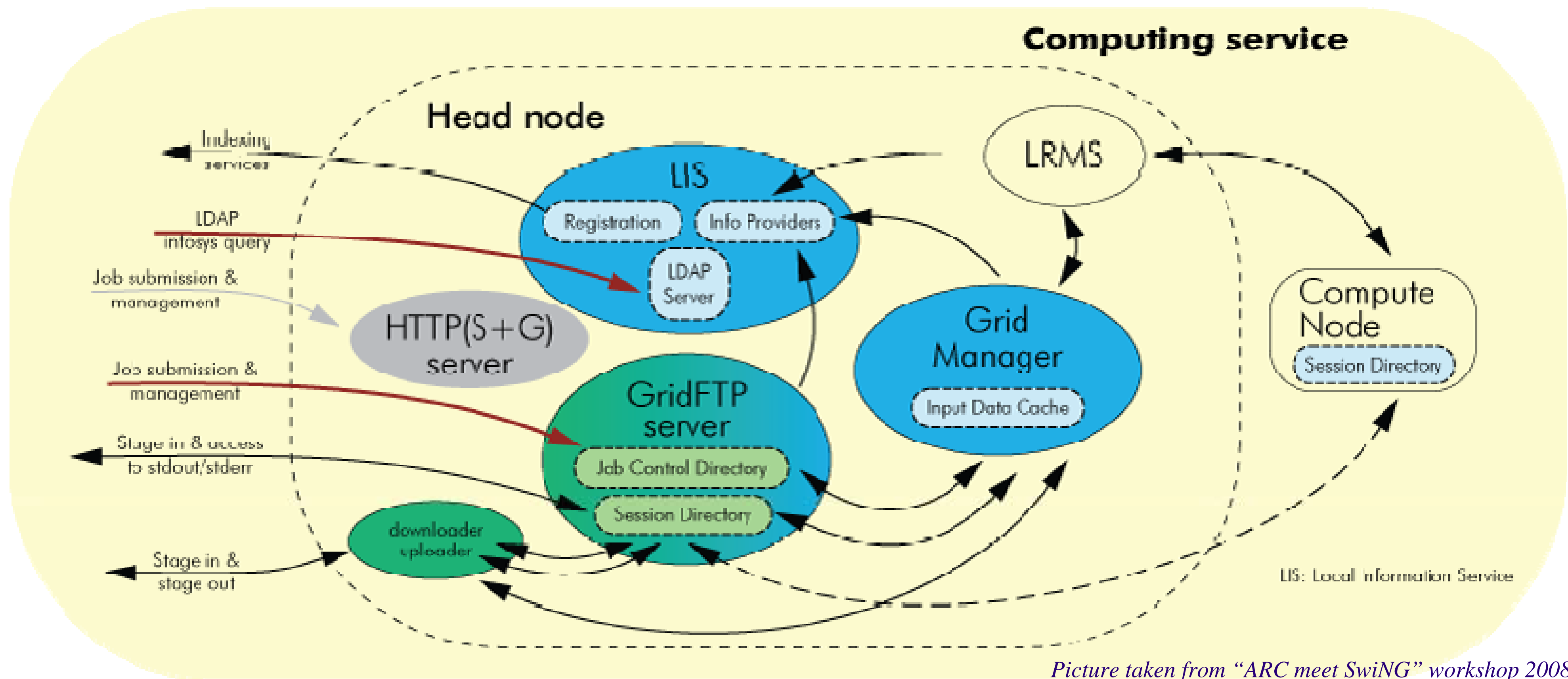
Sergio Maffioletti UZH/GC3
Marko Nikkimaki HES-SO
Sigve Haug UniBe/LHEP

ARC for sysadmins.
the tutorial

Sergio Maffioletti UZH/GC3
Marko Nikkimaki HES-SO
Sigve Haug UniBe/LHEP

Inside ARC: Client tools

Sergio Maffioletti
Grid Computing Competence Center GC3
University of Zurich
sergio.maffioletti@gc3.uzh.ch



Picture taken from "ARC meet SwiNG" workshop 2008

- Computing resources: Grid-enabled via ARC layer on head node (front-end):
  - Custom GridFTP server for all the communications
  - Grid Manager handles job management upon client request, interfaces to LRMS
  - Performs most data movement (stage in and out), cache management
  - Publishes resource and job information via LDAP

Lightweight User Interface with the built-in Resource Broker

- A set of command line utilities
- Minimal and simple
- Under the hood: resource discovery, matchmaking, optimization, job submission
- Complete support for single job management
- Basic functionality for multiple job management
- Built upon ARCLIB

Standalone binary client package possible to be installed in user space

1. Obtain access to a User Interface (ARC client software)
2. Request a user certificate from a Certification Authority
3. Deploy the signed certificate on the User Interface
4. Create grid proxy
5. Write a job description
6. Submit job
7. Monitor the progress of the job
8. Fetch the results

arcproxy – handle grid/voms proxy creation

arcsub – find suitable resources and submit a job

arcstat – check the status of jobs and resources

arccat – display stdout, stderr of a running job

arcget – retrieve the results of a finished job

arckill – stop a job

arcclean – delete a job from a computing resource

arcsync – find user's jobs

arcls – list files on a storage resource or in job's sandbox

arccp – transfer files to and from cluster and storages

- Create proxy: arcproxy
- Writing a job descrption: job.xrsl
- Submitting the job: arcsub
- Checking the status: arcstat/arccat
- Retrieving the result files: arcget

- Required to submit jobs to ARC
- Could be downloaded from http://ftp/nordugrid.org/download
- Various binary packages as well as source code
- Standalone package vailable for installing ARC client in user space

- Binaries available for system-wide installation

```
# yum install nordugrid-arc-client
```

The default behaviour of an ARC client can be con
gured by specifying alternative values for some
parameters in the client con
guration
le. The
le is called client.conf and is located in directory .arc
in user's home area:

`$HOME/.arc/client.conf`

SwiNG

- Resource Specification Language (RSL) files are used to specify job requirements and parameters for submission
  - ARC uses an extended language (xRSL) based on the Globus RSL
- Similar to scripts for local queuing systems, but includes some additional attributes
  - Job name
  - Executable location and parameters
  - Runtime Environment requirements

- helloWorld.sh

```
#!/bin/sh
echo “Hello World”
```

- helloWorld.xrsl

```
 & (executable=helloWorld.sh)
   (jobname=hellogrid)
   (stdout=std.out)
   (stderr=std.err)
   (gmlog=gridlog)
   (architecture=i686)
   (cputime=10)
   (memory=32)
```

SwiNG

Submit the job

```
# arcsub -d DEBUG -c ARC0:aio.grid.zoo -f helloWorld.xrsl
# => Job submitted with jobid
   gsiftp://aio.grid.zoo:2811/jobs/455611239779372141331307
```

- Query the status of the submitted job

```
# arcstat hellogrid
# Job gsiftp://aio.grid.zoo:2811/jobs/455611239779372141331307
  Jobname: hellogrid
  Status: INLRMS:Q
```

- Most common status values are: ACCEPTED, PREPARING, SUBMITTING, INLRMS:Q, INLRMS:R, EXECUTED, FINISHED

SwiNG

- Print the job output

```
# arccat hellogrid
```

  - Shows the standard output of the job
  - This can be done also during job execution

- Fetch the results

```
# arcget hellogrid
arcget: downloading files to
   /home/theuser/results/45561123977937214133107
arcget: download successful - deleting job from
   gatekeeper.
```

SwiNG

Inside ARC:
Runtime Environments

Sergio Maffioletti
Grid Computing Competence Center GC3
University of Zurich
sergio.maffioletti@gc3.uzh.ch

- Software packages which are pre-installed on a computing resource and made available through ARC
- Avoid the need of sending the binaries together with the job
- Allows local platform specific optimization
- Provides to the users a common environment for the specific application
- Implemented by shell scripts which initialize the environment and are placed in specific directory
- Required RTE can be specified in the job description file: (runtimeenvironment=APPS/LIFE/TANDEM-09.08)
- Every infrastructure should provide a registry for the supported RTEs and the conventions followed

Deployment and RTE: APPS/LIFE/TANDEM-09.08

```
..
export TANDEM LOCATION=$application base path
Export TANDEM_TAXONOMY=$TANDEM_LOCATION/bin

# Set the specific mdrun commands for this system.
export TANDEMRUN="$TANDEM_LOCATION/bin/tandem.exe”
..
```

SwiNG

In xrsl job description file

```
(runtimeenvironment="APPS/LIFE/TANDEM-09.08")
```

Within job execution

```
..
$TANDEMRUN input.xml
..
```

SwiNG

Inside ARC:
Sysadmin tips

Sergio Maffioletti
Grid Computing Competence Center GC3
University of Zurich
sergio.maffioletti@gc3.uzh.ch

SwiNG

Installation of ARC packages:

- For most rpm-based Linux distributions, RPMs for ARC and for most of its dependencies are provide through nordugrid repository
- Possible to install via apt or yum

```
# yum install nordugrid-arc-compute-element
```

- Provided deb packages

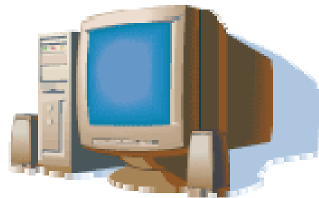
SwiNG

- ARC can be see as made of four main service type:
- ARC_CE: interface with the computing farm
- ARC_UI: client interface
- ARC_SE: interface with the storage farm
- ARC_GIIS: top level information system

- Each of them can be installed either separately or altogether on the same node
- RPMs are provided for ARC server and ARC client
- ARC server includes components for CE,SE,GIIS
- System administrator decides which service configure and enable through configuration files

- Given a computing farm controlled and managed by a Local Resource Management System (LRMS)
- ARC_CE is the interface to the LRMS
- ARC_CE needs to be an authorized client of the LRMS
- ARC_CE needs to share at least one filesystem with the rest of the computing farm
- Submission to the LRMS is done by ARC_CE on behalf of the users
- ARC_CE checks the status of the LRMS jobs and retrieves the results on behalf of the user
- Results form the LRMS submission are stored on ARC_CE for manual retrieval or transfer to a storage resource

Nordugrid Client



Resource selection

- ARC_UI embodies a resource broker that is responsible of selecting the resources to match the requirements of a submitting job
- Broker first queries the GIIS it knows to get a list of sites
- Then queries the sites to check whether the user is authorized to the site
- Then filters the resources according to the ARC_job's resource specifications
- Then ranks the filtered resources according to its policy (random, fastest cpus, …)
- The top rank resource is selected
- Submission to selected resource

- An ARC_job is submitted from ARC_UI
- On ARC_CE, the Gridftp server accept the request
- Authentication and authorization (GSI,VOMS)
  - Request is mapped to local user account
- An ARC_jobID is created (this will be the unique reference for the job)
- A session folder is created within $sessiondir (as specified in arc.conf) named as the ARC_jobID
- Downloader process is started to fetch input data
- Input data are stored in ARC_job's session dir
- submit-$LRMS-job script is started to translate ARC_job into a local submission
  - There are several LRMS backend: PBS, SGE, LL, LSF, Condor,…

SwiNG

- Translated job is submitted to LRMS using local user account
- Lifecycle of LRMS_job is supervised by grid-manager
  - It executes scripts like: scan-$LRMS-job
- Information system updates information on the status of the job (INLRMS:R means submitted to LRMS and running there)
- Once LRMS_job is terminated, uploader process takes care of staging results to a designated storage resource (if specified in xrsl)
- ARC_job status is reported as FINISHED

SwiNG

- Location of log files can be specified in arc.conf
  - Each service has its own section where individual log can be configured (location, rotation policy, verbosity level)

SwiNG

default configuration:

log location: /var/log/arc/gridftpd.log

control: /etc/init.d/gridftpd [start, status, stop]

daemon: gridftpd

Open port on: 2811 (default)

FTP PASS mode: 9000 – 9500 (default)

default configuration:

log location: /var/log/arc/grid-manager.log

control: /etc/init.d/a-rex [start, status, stop]

daemon: arched

open port on: 443 (when Web Service Interface activated)

handles $controldir (/var/spool/nordugrid/jobstatus)

uses several perl and bash scripts located in /usr/shar/arc

submit-$LRMS-job, scan-$LRMS-job, ...

default configuration:

log location:

- /var/log/arc/infoprovider.log
- /var/log/arc/inforegistration.log
- /var/log/arc/bdii/bdii-update.log
- /var/log/bdii/bdii.log

control: /etc/init.d/grid-infosys [start, status, stop]

Daemons: slapd, ldapadd, bdii-fw

open port: 2135

Ldif files: var/run/arc /var/run/bdii

SwiNG

SwiNG

ARC for sysadmins. The tutorial