



Enabling Grids for E-science

Submission, Monitoring and Control of Jobs

José Luis Vázquez-Poletti (UCM)

*Training the Trainers (CERN)
16-18 June 2008*

www.eu-egEE.org

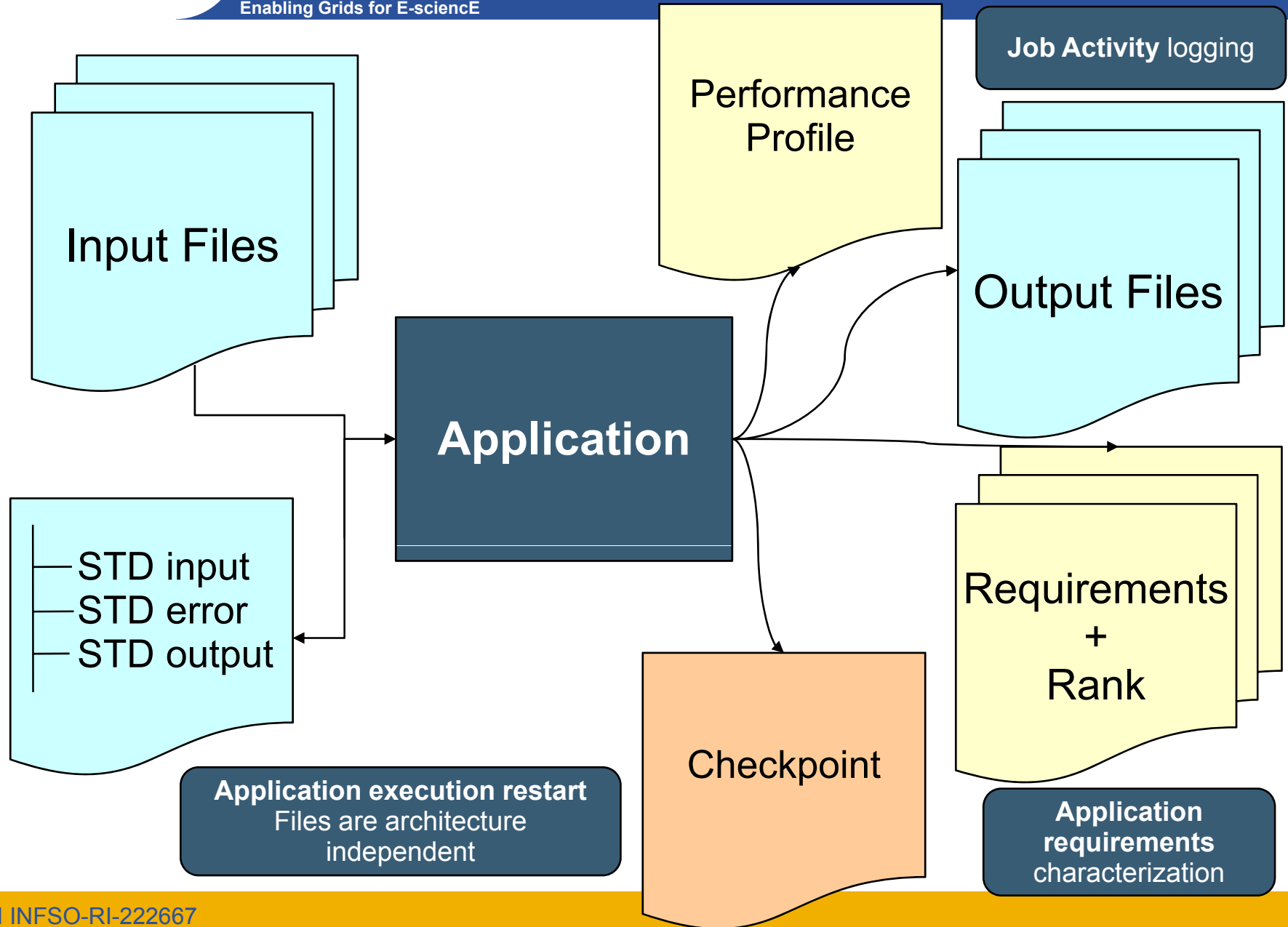


Information Society
and Media

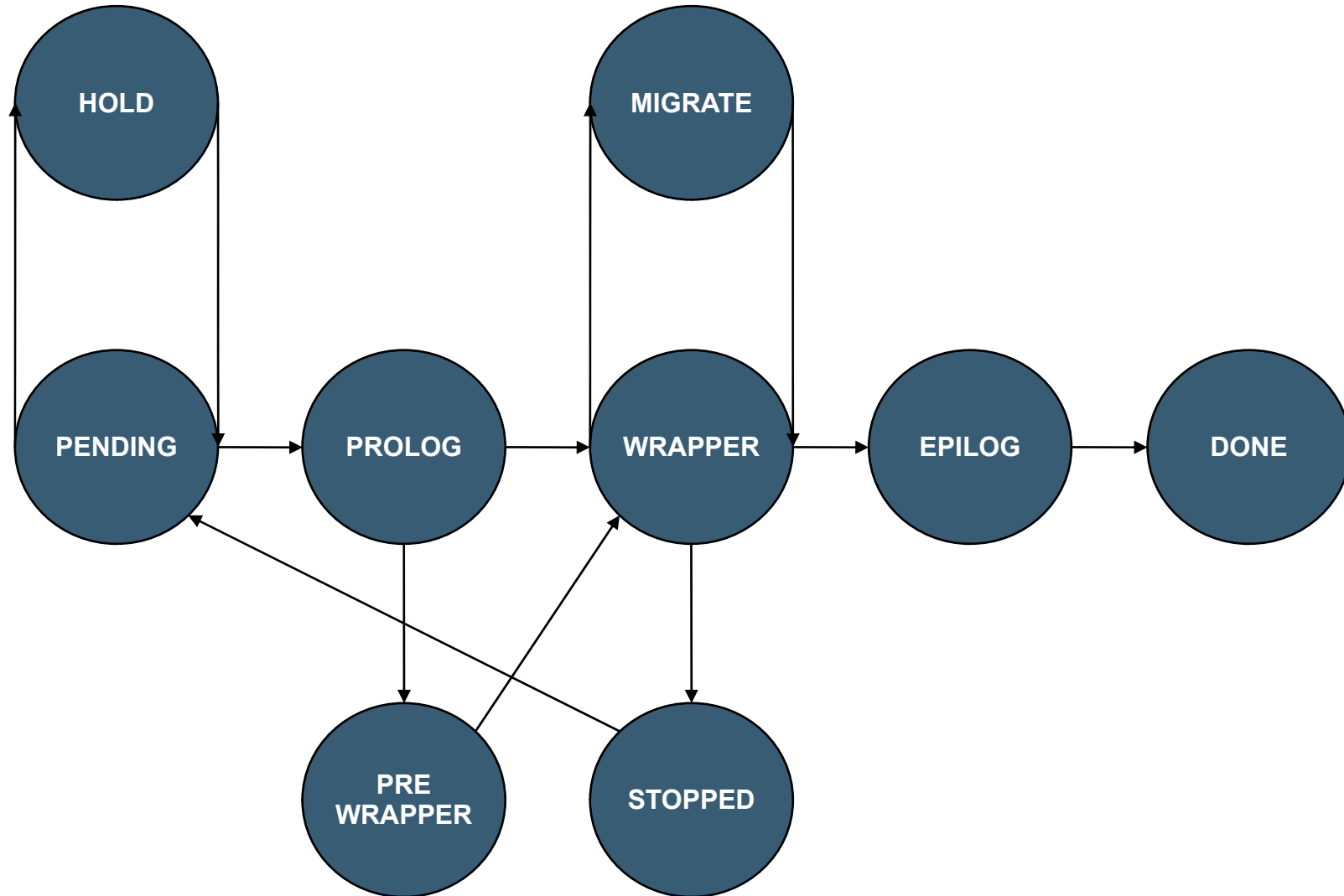


- 1. User Model Overview**
2. Usage Scenarios
3. Job Definition
4. Commands in detail

User Model Overview



Life-cycle



Main Commands

- **gwps**: Shows job information and state
- **gwhistory**: Shows execution history
- **gckill**: Sends signals to a job (kill, stop, resume, reschedule)
- **gws submit**: Submits a job or array
- **gwwait**: Waits for job's end (any, all, set)
- **gwuser**: User Monitoring
- **gwhost**: Host Monitoring
- **gwacct**: Accounting

1. User Model Overview
- 2. Usage Scenarios**
3. Job Definition
4. Commands in detail

Single Job

- Create your proxy.
- Create a simple Job Template:

```
EXECUTABLE = /bin/ls
```

- and save it as **jt** in directory example.
- Use *gws submit* command to submit the job:

```
$ gws submit -t example/jt
```

- Use *gwhost* command to see available resources:

HID	PRI	OS	ARCH	MHZ	%CPU	MEM(F/T)	DISK(F/T)	N(U/F/T)	LRMS	HOSTNAME
0	1	Linux2.6.17-2-6	x86	3216	0	44/2027	76742/118812	0/0/2	Fork	cygnus.dacya.ucm.es
1	1			0	0	0/0	0/0	0/0/0		orion.dacya.ucm.es
2	1	Linux2.6.18-4-a	x86_6	2211	100	819/1003	77083/77844	0/2/4	PBS	hydrus.dacya.ucm.es
3	1	Linux2.6.17-2-6	x86	3216	163	1393/2027	101257/118812	0/2/2	Fork	draco.dacya.ucm.es
4	1	Linux2.6.18-4-a	x86_6	2211	66	943/1003	72485/77844	0/5/5	SGE	aquila.dacya.ucm.es

- and get more detailed information specifying a Host ID:

```
$ gwhost 0
```

HID	PRI	OS	ARCH	MHZ	%CPU	MEM(F/T)	DISK(F/T)	N(U/F/T)	LRMS	HOSTNAME
0	1	Linux2.6.17-2-6	x86	3216	0	50/2027	76393/118812	0/0/2	Fork	cygnus.dacya.ucm.es

QUEUE	SL(F/T)	WALLT	CPUT	COUNT	MAXR	MAXQ	STATUS	DISPATCH	PRIORITY
default	0/2	0	-1	0	-1	0	enabled	NULL	0

- Check the resources that match job requirements with `gwhost -m 0`:

```
$ gwhost -m 0
HID QNAME      RANK  PRIO  SLOTS  HOSTNAME
0   default    0     1     0      cygnus.dacya.ucm.es
2   default    0     1     3      hydrus.dacya.ucm.es
2   qlong      0     1     3      hydrus.dacya.ucm.es
2   qsmall    0     1     3      hydrus.dacya.ucm.es
3   default    0     1     0      draco.dacya.ucm.es

4   all.q      0     1     3      aquila.dacya.ucm.es
```

- Follow the evolution of the job with `gwps` command:

```
$ gwps
USER          JID DM   EM   START      END      EXEC      XFER      EXIT NAME  HOST
gwtutorial00  0   done ---- 20:16:28 20:18:16 0:00:55 0:00:08 0   stdin aquila.dacya.ucm.es/SGE
tinova        1   done ---- 12:26:46 12:31:15 0:03:55 0:00:08 0   stdin hydrus.dacya.ucm.es/PBS

tinova        2   pend ---- 12:38:38 ---:---:--- 0:00:00 0:00:00 --   t.jt  --
```

- HINT:** Use `gwps -c <seconds>` for continuous output.

Single Job

- See the job history with *gwhistory* command:

```
$ gwhistory 4
HID START      END          PROLOG WRAPPER EPILOG  MIGR    REASON QUEUE    HOST
2    12:58:04 12:58:16 0:00:06 0:00:04 0:00:02 0:00:00 ----  default  hydrus.dacya.ucm.es/PBS
```

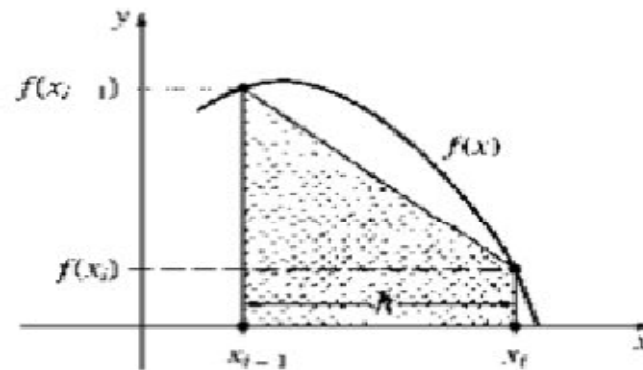
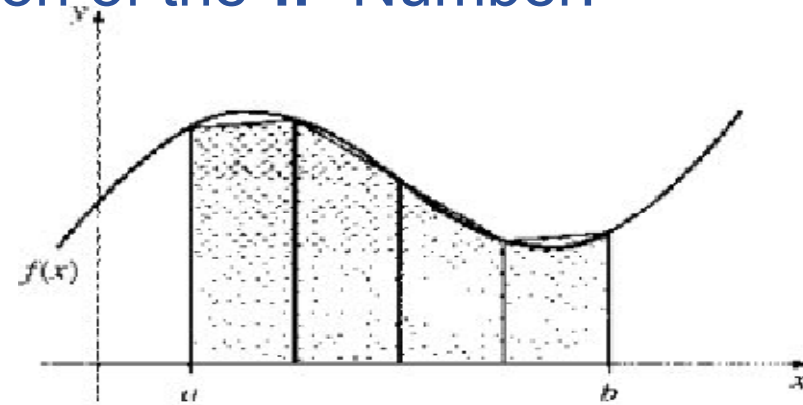
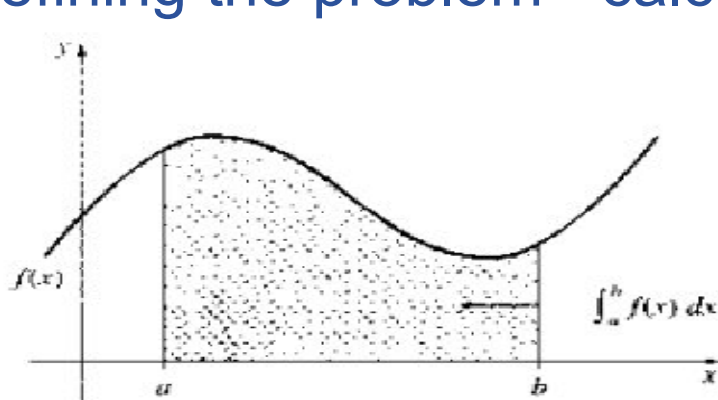
- Once finished... time to retrieve the results:

```
$ ls -lt stderr.4 stdout.4
-rw-r--r-- 1 tinova tinova 0 2007-09-07 12:58 stderr.4
-rw-r--r-- 1 tinova tinova 72 2007-09-07 12:58 stdout.4

$ cat stdout.4
job.env
stderr.execution
stderr.wrapper
stdout.execution
stdout.wrapper
```

Array Jobs

- Defining the problem - calculation of the π Number:



- pi.c calculates each slice:

Examples Directory:

\$GW_LOCATION/share/examples/

```
#include <string.h>
#include <stdlib.h>

int main (int argc, char** args)
{
    int task_id;
    int total_tasks;
    long long int n;
    long long int i;

    double l_sum, x, h;

    task_id = atoi(args[1]);
    total_tasks = atoi(args[2]);
    n = atoll(args[3]);

    fprintf(stderr, "task_id=%d total_tasks=%d n=%lld\n", task_id,
total_tasks, n);

    h = 1.0/n;

    l_sum = 0.0;

    for (i = task_id; i < n; i += total_tasks)
    {
        x = (i + 0.5)*h;
        l_sum += 4.0/(1.0 + x*x);
    }

    l_sum *= h;

    printf("%0.12g\n", l_sum);

    return 0;
}
```



```
$ gcc -O3 pi.c -o pi
```

pi arguments:

- Task ID
- Total tasks
- Integral intervals

Array Jobs

- Create a job template (pi.jt):

```
EXECUTABLE = pi
ARGUMENTS = $(TASK_ID) $(TOTAL_TASKS) 100000
STDOUT_FILE = stdout_file.$(TASK_ID)
STDERR_FILE = stderr_file.$(TASK_ID)
RANK = CPU_MHZ
```

- Submit the array of jobs:

```
$ gsubit -v -t pi.jt -n 4
ARRAY ID: 0

TASK JOB
0      3
1      4
2      5
3      6
```

- Use the **gwait** command to wait for the jobs:

```
$ gwait -v -A 0
0      : 0
1      : 0
2      : 0
3      : 0
```

Array Jobs

- At the end we have the following STDOUT files:

```
stdout_file.0  
stdout_file.1  
stdout_file.2  
stdout_file.3
```

- Sum the contained values to get the value of π :

```
$ awk 'BEGIN {sum=0} {sum+=$1} END {printf "Pi is %0.12g\n", sum}' stdout_file.*  
Pi is 3.1415926536
```

- IDEA: Embedding all in script? Check the examples directory ...

Workflow Jobs

- GridWay can handle workflows with the following functionality:
 - Sequence, parallelism, branching and looping structures
 - The workflow can be described in an abstract form without referring to specific resources for task execution
 - Quality of service constraints and fault tolerance are defined at task level
- Job dependencies specified by using the *-d* option of the *gws submit* command

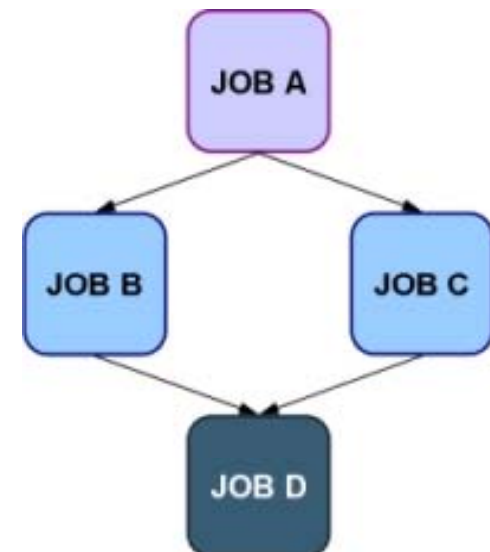
```

• $ gws submit -v -t A.jt
  JOB ID: 5

• $ gws submit -v -t B.jt -d "5"
  JOB ID: 6

• $ gws submit -v -t C.jt -d "5"
  JOB ID: 7

• $ gws submit -t D.jt -d "6 7"
    
```



1. User Model Overview
2. Usage Scenarios
- 3. Job Definition**
4. Commands in detail

Job Template

Generic

- NAME = Name of the job.

Execution

- EXECUTABLE = Executable file.
- ARGUMENTS = Arguments for the executable.
- ENVIRONMENT = User defined, comma-separated, environment variables.
- TYPE = “Single”, “multiple” and “mpi” (like GRAM).
- NP = Number of processors in MPI jobs.

I/O Files

- INPUT_FILES = A comma-separated pair of “local remote” filenames.
- OUTPUT_FILES = A comma-separated pair of “remote local” filenames.

Standard Streams

- `STDIN_FILE` = Standard Input file.
- `STDOUT_FILE` = Standard Output file.
- `STDERR_FILE` = Standard Error file.

Check pointing

- `RESTART_FILES` = Checkpoint files, architecture independent.
- `CHECKPOINT_INTERVAL` = Seconds for checkpoint files transfer.
- `CHECKPOINT_URL` = GridFTP URL to store checkpoint files.

Resource Selection

- `REQUIREMENTS` = Boolean expression. If true, host will be considered for scheduling.
- `RANK` = Numerical expression evaluated for each host considered for scheduling.

Scheduling

- RESCHEDULING_INTERVAL = How often GridWay searches better resources for the job.
- RESCHEDULING_THRESHOLD = Migration will occur when a better resource is discovered and job is running less than this threshold.
- DEADLINE = Deadline of job start.

Performance

- SUSPENSION_TIMEOUT = Max suspension time in local job management system.
- CPULOAD_THRESHOLD = Load threshold for the CPU assigned to job.
- MONITOR = Optional program to monitor job performance.

Fault Tolerance

- RESCHEDULE_ON_FAILURE = Behaviour in case of failure.
- NUMBER_OF_RETRIES = Retries in case of failure.

Advanced Job Execution

- WRAPPER = Script for wrapper.
- PRE_WRAPPER = Optional program to be executed before the actual job (i.e. additional remote setup).
- PRE_WRAPPER_ARGUMENTS = Arguments for pre-wrapper program.

I/O Files

- General Syntax: SRC1 DST1, SRC2 DST2,...
- Absolute path: EXECUTABLE = /bin/ls
- GridFTP URL: INPUT_FILES =
gsiftp://machine/tmp/input_exp1 input
- File URL: INPUT_FILES = file:///etc/passwd
- Name: INPUT_FILES = test_case.bin
 - NOTE: The source names for output files MUST be a single name, do not use absolute paths or URLs

Standard Streams

- Any of the above methods except:
 - STDIN_FILE : Cannot specify a destination name
 - {STDOUT, STDERR}_FILE : Cannot specify a source name (only destination)

Generics

- Variables can be used in the value string of each option
 - *with the format: $\${GW_VARIABLE}$*
- These variables are substituted at run time with its corresponding value.
 - *For example: $STDOUT_FILE = stdout.\${JOB_ID}$*

Valid Variables

- $\${JOB_ID}$ Job ID.
- $\${ARRAY_ID}$ Job array ID. -1 if job is not in any.
- $\${TASK_ID}$ Task ID within job array. -1 if job is not in any.
- $\${ARCH}$ Architecture of selected execution hosts.
- $\${PARAM}$ Allows assignment of arbitrary start and increment values for array jobs (e.g. file naming patterns).
- $\${MAX_PARAM}$ Upper bound for the $\${PARAM}$ variable.

Resource Selection

- Two variables can be used to define valid resources for a given job.
 - **REQUIREMENTS:** Express conditions that *BAN* resources
 - **RANK:** Express conditions over the *PREFERENCE* of resources

```

stmt ::= expr ';'
expr ::= VARIABLE '=' INTEGER
      | VARIABLE '>' INTEGER
      | VARIABLE '<' INTEGER
      | VARIABLE '=' STRING
      | expr '&' expr
      | expr '|' expr
      | '!' expr
      | '(' expr ')'
    
```

Requirements

```

stmt ::= expr ';'
expr ::= VARIABLE
      | INTEGER
      | expr '+' expr
      | expr '-' expr
      | expr '*' expr
      | expr '/' expr
      | '-' expr
      | '(' expr ')'
    
```

Rank

Resource Selection

- **HOSTNAME** – FQDN.
- **ARCH** – Architecture of execution host.
- **OS_NAME** – Operative System.
- **OS_VERSION** – Operative System version.
- **CPU_MODEL** – CPU model.
- **CPU_MHZ** – CPU speed in MHZ.
- **CPU_FREE** – Percentage of free CPU.
- **CPU_SMP** – CPU SMP size.
- **NODECOUNT** – Number of nodes.
- **SIZE_MEM_MB** – Memory size in MB.
- **FREE_MEM_MB** – Free memory in MB.
- **SIZE_DISK_MB** – Disk space in MB.

Resource Selection

- **FREE_DISK_MB** – Free disk space in MB.
- **LRMS_NAME** – Name of local DRM system.
- **LRMS_TYPE** – Type of local DRM system.
- **QUEUE_NAME** – Name of the queue.
- **QUEUE_NODECOUNT** – Number of queue nodes.
- **QUEUE_FREENODECOUNT** – Free queue nodes.
- **QUEUE_MAXTIME** – Max wall time for jobs in queue.
- **QUEUE_MAXCPU TIME** – Max CPU time of jobs in queue.
- **QUEUE_MAXCOUNT** – Max jobs that can be submitted in one request.
- **QUEUE_MAXRUNNINGJOBS** – Max running jobs in queue.
- **QUEUE_MAXJOBSINQUEUE** – Max queued jobs in queue.
- **QUEUE_DISPATCHTYPE** – Queue dispatch type.
- **QUEUE_PRIORITY** – Priority of queue.
- **QUEUE_STATUS** – Status of queue (i.e. “active”, “production”).

Job Environment

- Job environment variables can be set with the *ENVIRONMENT* parameter.
 - The variables defined in the *ENVIRONMENT* are "sourced" in a bash shell
 - **ENVIRONMENT = VAR = "expr \${JOB_ID} + 3" # will set VAR to JOB_ID + 3**
-
- | | |
|---|--|
| <ul style="list-style-type: none"> ▪ GW_RESTARTED ▪ GW_EXECUTABLE ▪ GW_ARCH ▪ GW_CPU_MHZ ▪ GW_MEM_MB ▪ GW_RESTART_FILES ▪ GW_CPULOAD_THRESHOLD ▪ GW_ARGUMENTS ▪ GW_TASK_ID ▪ GW_CPU_MODEL | <ul style="list-style-type: none"> ▪ GW_ARRAY_ID ▪ GW_TOTAL_TASKS ▪ GW_JOB_ID ▪ GW_OUTPUT_FILES ▪ GW_INPUT_FILES ▪ GW_OS_NAME ▪ GW_USER ▪ GW_DISK_MB ▪ GW_OS_VERSION |
|---|--|

1. User Model Overview
2. Usage Scenarios
3. Job Definition
4. **Commands in detail**

gws submit – submitting jobs

```
gws submit <-t template> [-n tasks] [-h] [-v] [-o] [-s start] \
                        [-i increment] [-d "id1 id2 ..."]
```

- **OPTIONS**

- **-h** - Prints help.
- **-t <template>** - The template file describing the job.
- **-n <tasks>** - Submit an array job with the given number of tasks.
 - All the jobs in the array will use the same template.
- **-s <start>** - Start value for custom param in array jobs. Default 0.
- **-i <increment>** - Increment value for custom param in array jobs
 - Each task has associated the value $PARAM=start+increment * TASK_ID$, and $MAX_PARAM = start+increment*(tasks-1)$. Default 1.
- **-d <"id1 id2...">** - Job dependencies.
 - Submit the job on hold state, and release it once jobs with id1,id2,.. have successfully finished.
- **-v** - Print to stdout the job ids returned by gwd.
- **-o** - Hold job on submission.
- **-p <priority>** - Initial priority for the job.

gwps – monitoring jobs

```
gwps [-h] [-u user] [-r host] [-A AID] [-s job_state] \
      [-o output_format] [-c delay] [-n] [job_id]
```

- **OPTIONS**

- **-h** - Prints help.
- **-u user** - Monitor only jobs owned by user.
- **-r host** - Monitor only jobs executed in host.
- **-A AID** - Monitor only jobs part of the array AID.
- **-s job_state** - Monitor only jobs in states matching that of job_state.
- **-o output_format** - Formats output information, allowing the selection of which fields to display.
- **-c <delay>** - This will cause gwps to print job information every <delay> seconds continuously (similar to top command).
- **-n** - Do not print the header.
- **job_id** - Only monitor this job_id.

gwhistory – accessing job history

```
gwhistory [-h] [-n] <job_id>
```

- **OPTIONS**

- **-h** - Prints help.
- **-n** - Do not print the header lines.
- **job_id** - Job identification as provided by gwps.

gwhost – monitoring hosts

```
gwhost [-h] [-c delay] [-nf] [-m job_id] [host_id]
```

- **OPTIONS**

- **-h** - Prints help.
- **-c <delay>** - This will cause gwhost to print job information every <delay> seconds continuously (similar to top command).
- **-n** - Do not print the header.
- **-f** - Full format.
- **-m <job_id>** - Prints hosts matching the requirements of a given job.
- **host_id** - Only monitor this host_id, also prints queue information

gkill – signalling jobs

```
gkill [-h] [-a] [-k | -t | -o | -s | -r | -l | -9] <job_id \
[job_id2 ...] | -A array_id>
```

- **OPTIONS**

- **-h** - Prints help.
- **-a** - Asynchronous signal, only relevant for KILL and STOP.
- **-k** - Kill (default, if no signal specified).
- **-t** - Stop job.
- **-r** - Resume job.
- **-o** - Hold job.
- **-l** - Release job.
- **-s** - Re-schedule job.
- **-9** - Hard kill, removes the job from the system without synchronizing remote job execution or cleaning remote host.
- **job_id [job_id2 ...]** - Job identification as provided by gwps. You can specify a blank space separated list of job ids.
- **-A <array_id>** - Array identification as provided by gwps.

gwwait – waiting for jobs

```
gwwait [-h] [-a] [-v] [-k] <job_id...| -A array_id>
```

- **OPTIONS**

- **-h** - Prints help.
- **-a** - Any, returns when the first job of the list or array finishes.
- **-v** - Prints job exit code.
- **-k** - Keep jobs, they remain in fail or done states in the GridWay system.
 - By default, jobs are killed and their resources freed.
- **-A <array_id>** - Array identification as provided by gwps.
- **job_id ...** - Job ids list (blank space separated).

gwuser – accessing user information

```
gwuser [-h] [-n]
```

- **OPTIONS**
 - **-h** - Prints help.
 - **-n** - Do not print the header.

gwacct – accessing accounting information

```
gwacct [-h] [-n] [<-d n | -w n | -m n | -t s>] \  

      <-u user|-r host>
```

- **OPTIONS**

- **-h** - Prints help.
- **-n** - Do not print the header.
- **<-d n | -w n | -m n | -t s>** - Take into account jobs submitted after certain date
 - specified in number of days (-d), weeks (-w), months (-m) or an epoch (-t).
- **-u user** - Print usage statistics for user.
- **-r hostname** - Print usage statistics for host.

**Thank you
for your attention!**