



Submission, Monitoring and Control of Jobs

OGF25
Catania, Italy
March 2nd, 2009

Alejandro Lorca

**Distributed Systems Architecture Group
Universidad Complutense de Madrid**



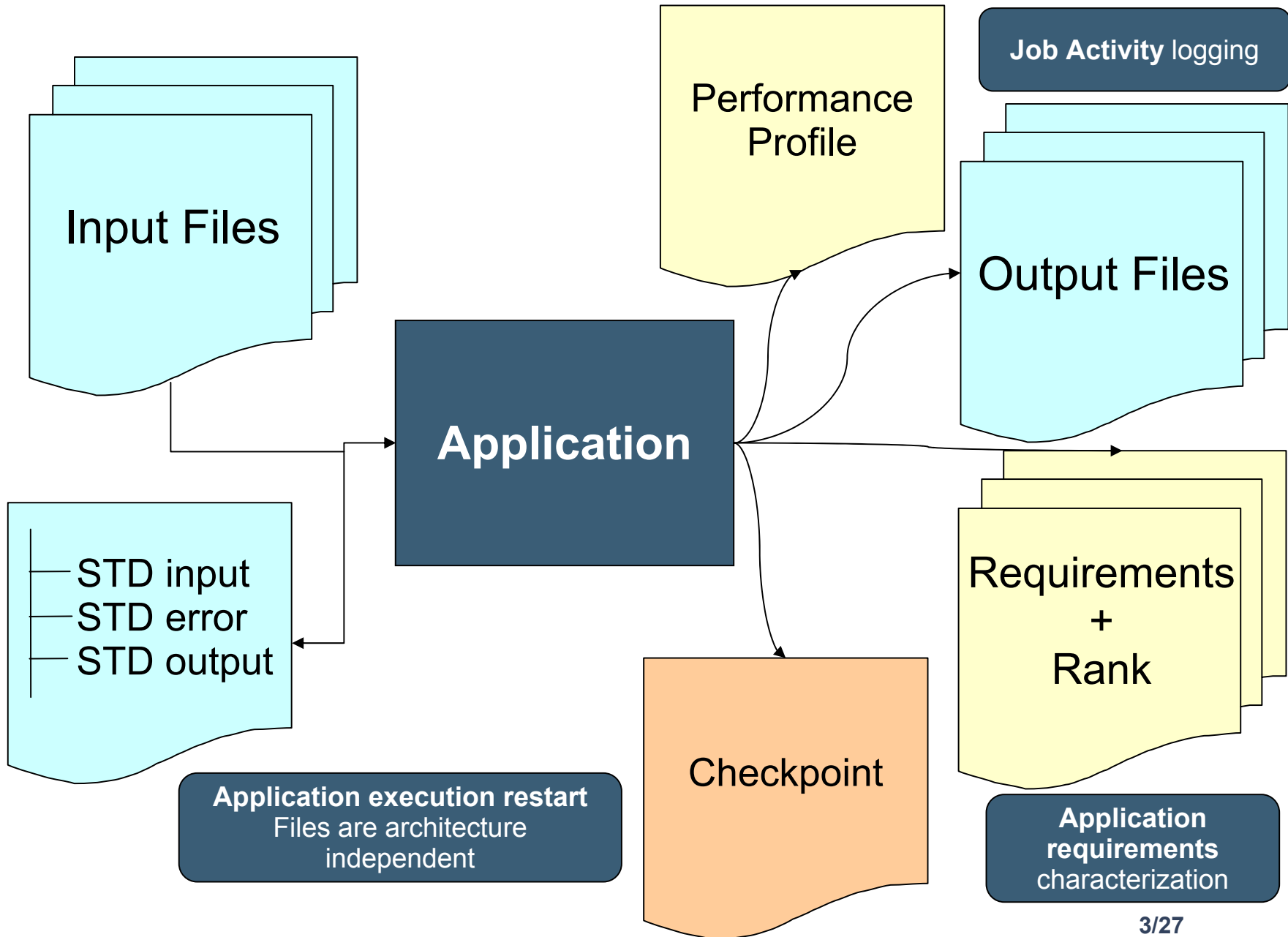


- **User Model Overview**
- Usage Scenarios
- Job Definition



User Model Overview

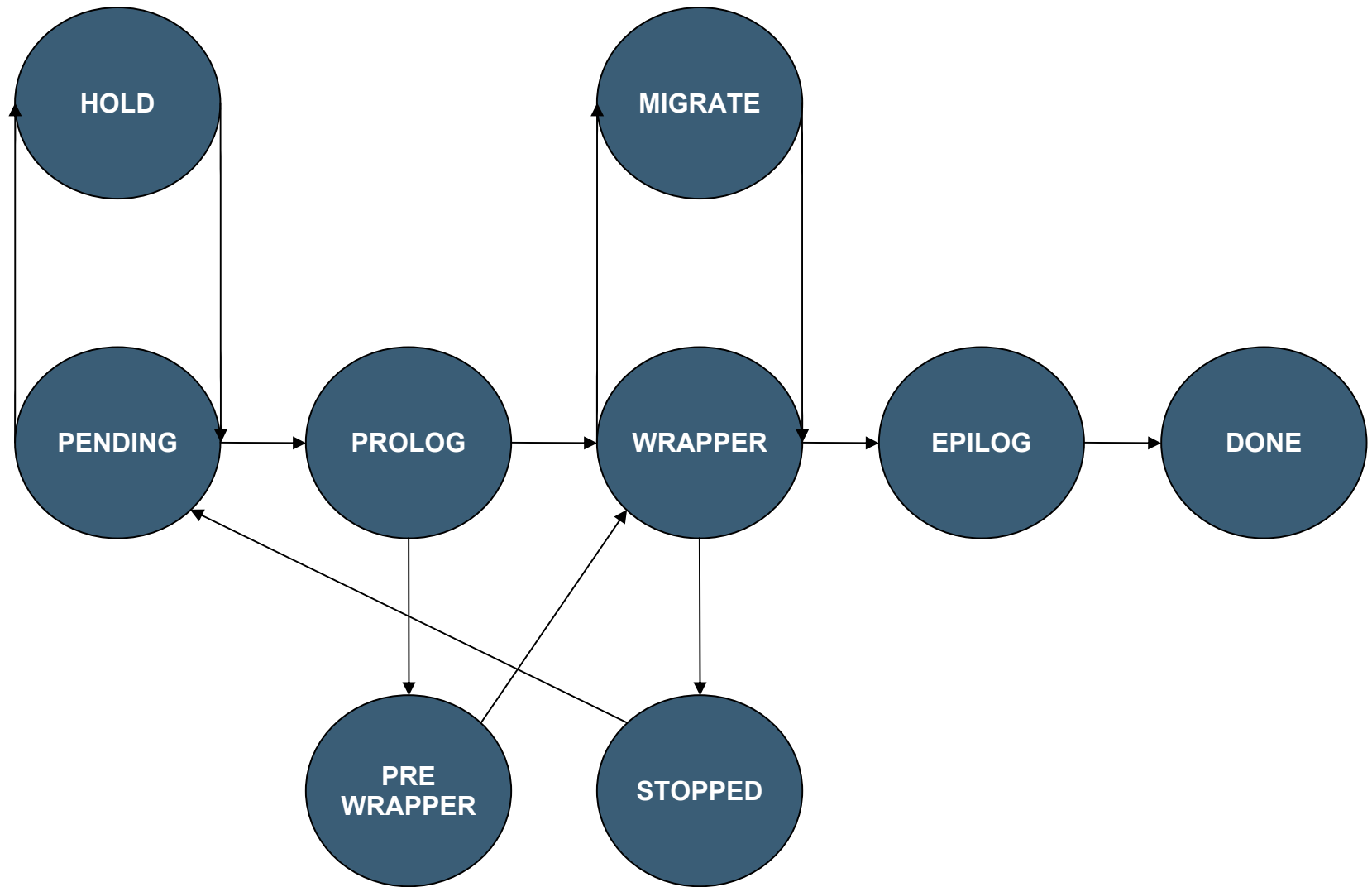
dsa-research.org





User Model Overview

Life-cycle





User Model Overview

Main Commands

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



- User Model Overview
- **Usage Scenarios**
- Job Definition



Usage Scenarios

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



Usage Scenarios

Single Job

- 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:

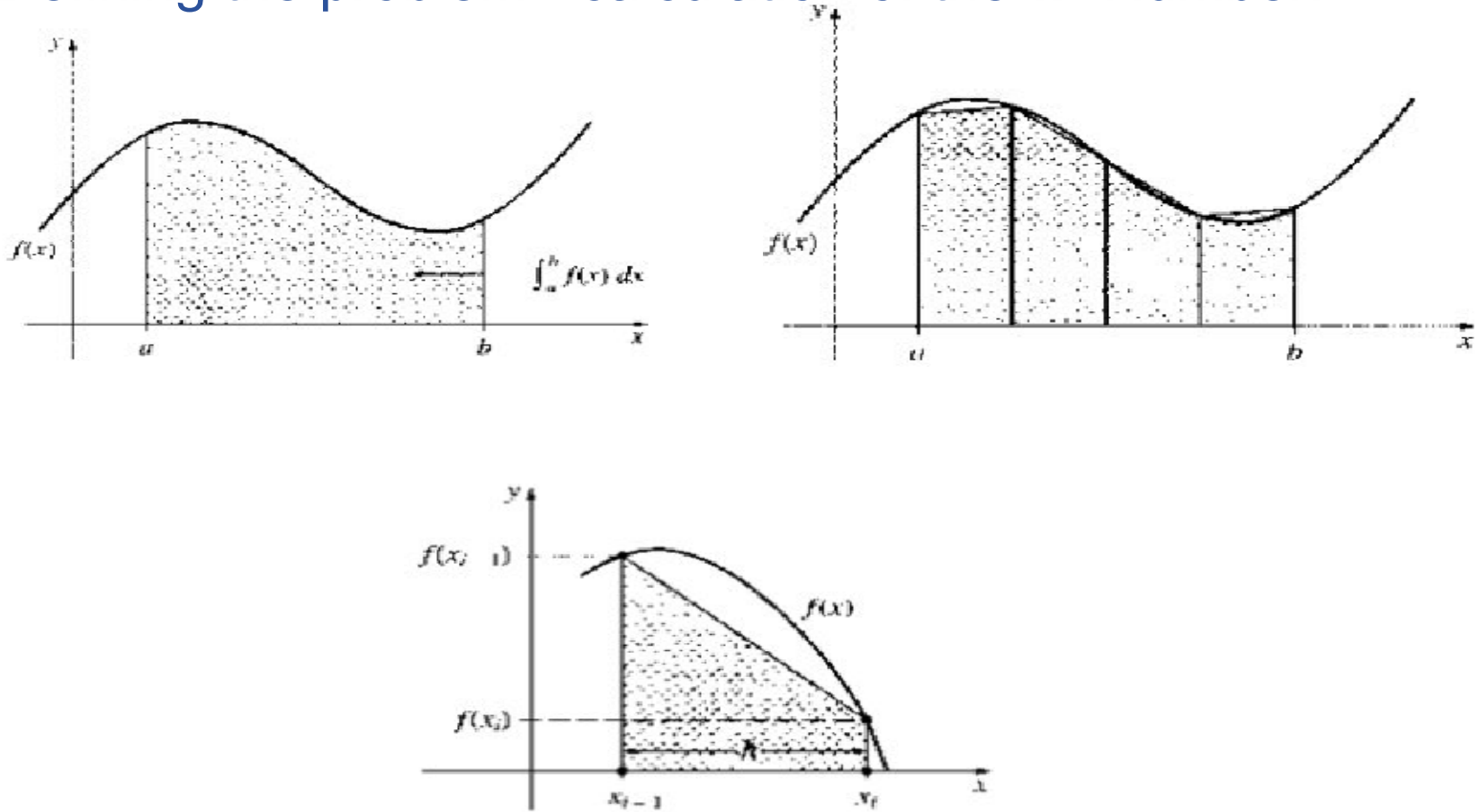
```
$ ls -lt stderr.4 stdout.4
-rw-r--r-- 1 tinova tinova 0 2007-09-07 12:58 stderr.4
-rw-r--r-- 1 tinova tinova 72 2007-09-07 12:58 stdout.4

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

Usage Scenarios

Array Jobs

- Defining the problem - calculation of the π Number:





Usage Scenarios

• 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("%.12g\n", l_sum);

    return 0;
}
```

IMPORTANT
32bits resources: -m32

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

pi arguments:
• Task ID
• Total tasks
• Integral intervals
```

dsa-research.org

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:

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

TASK	JOB
0	3
1	4
2	5
3	6

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

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



Usage Scenarios

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 :

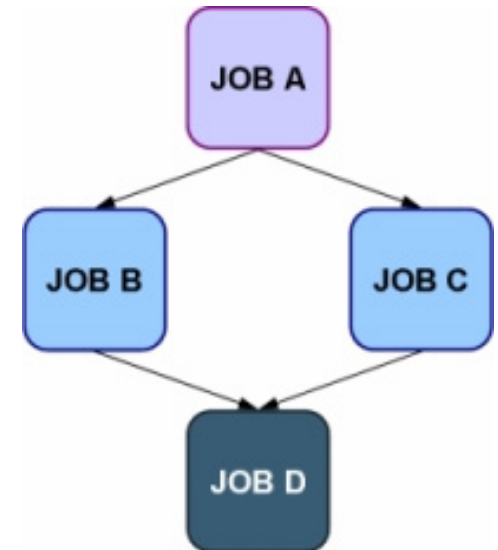
```
$ awk 'BEGIN {sum=0} {sum+=$1} END {printf "Pi is %0.12g\n", sum}' stdout_file.*  
Pi is 3.1415926536
```

- 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"
```





- User Model Overview
- Usage Scenarios
- **Job Definition**



Job Definition

Job Template

Generic

- NAME = Name of the job.

Execution

- EXECUTABLE = Executable file.
- ARGUMENTS = Arguments for the executable.
- REQUIREMENTS = (OPTIONAL) Filter condition

I/O Files

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

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.

Job Definition

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



Job Definition

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.



**Thank you
for your attention!**