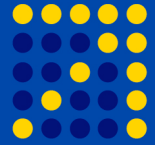


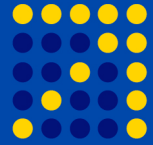
Tools to run Bio Application on the gLite grid infrastructure: JST and Parrot

G. Donvito
INFN-BARI



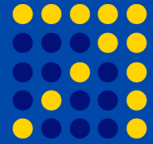


- Introduction to problems
- Issues
- Job Submission Tool
- Parrot
- Conclusions

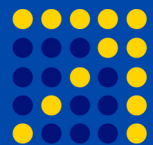


Introduction to problems

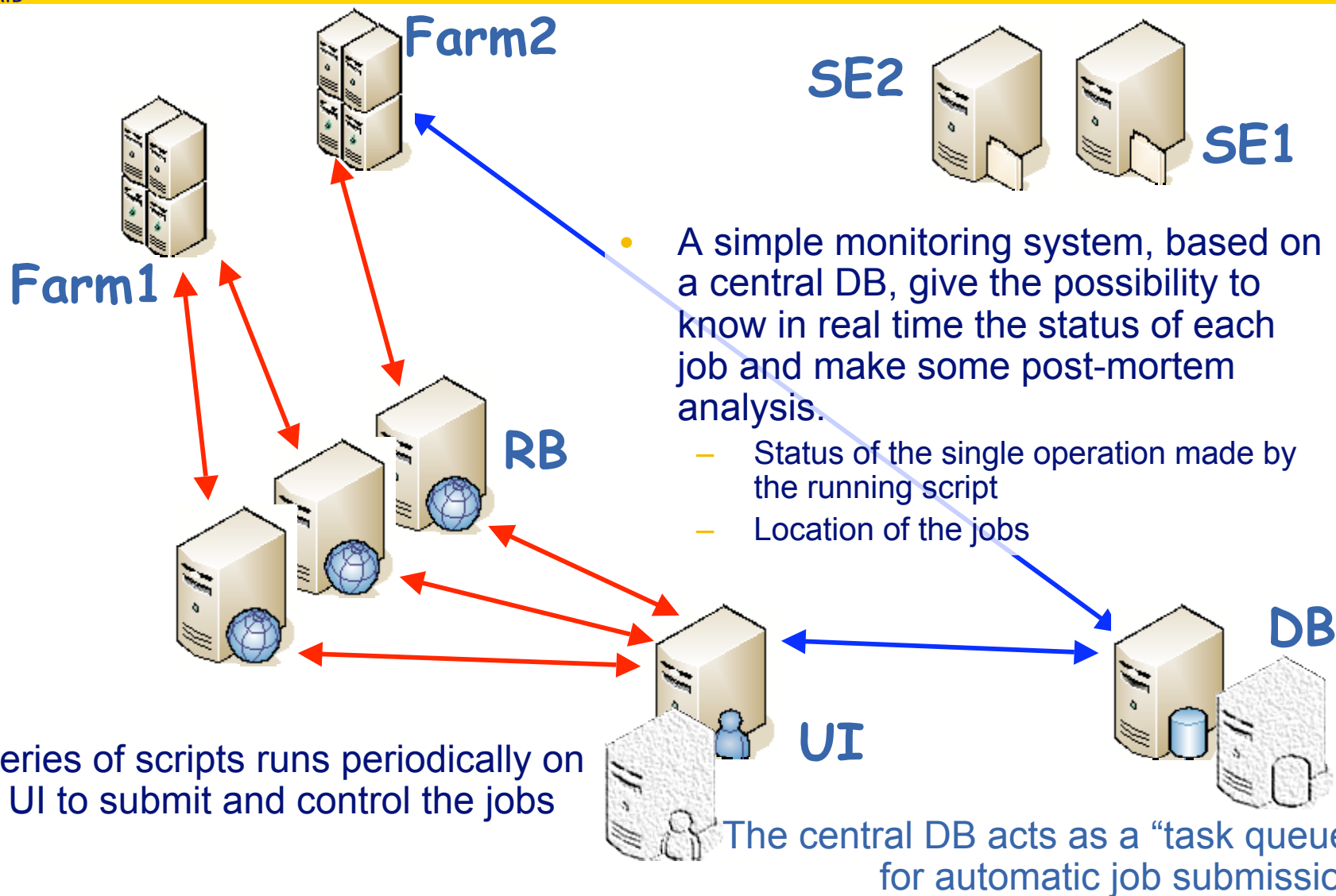
- Usual problems with bioinformatics applications:
 - Access data
 - Usually application are “data bound”
 - Often it is difficult or impossible to change the code of the application in order to exploit gLite features
 - Run a huge amount of “tasks”.
 - Genome analysis typically means e huge number of “jobs” of the same executable with different input files or parameters
 - It difficult to manage this runs manually, dealing with failures, resubmission, etc.

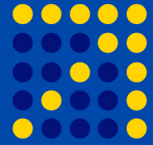


- How to manage the required huge amount of “jobs”?
- How to enable application to access files from the grid?
- It is necessary
 - To keep trace of the tasks executed, failed or running
 - The failed task must be automatically re-submitted
 - To know the current “status” of a task
 - To run it in a unattended way
 - To avoid “single-point“ of failure (that can stop the running)
 - To access the data in a transparent way (without knowing their location) possibly without moving them
- We have developed
 - a general purpose “Job Submission Tool”
 - a procedure to access transparently the data, independently from their physical location on the Grid



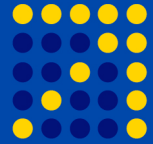
JOB SUBMISSION SCHEMA





Generic “Job Submission Tool”

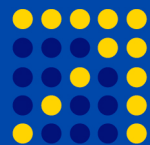
- **Identical Jobs** get submitted to grid
- The Jobs are then instructed about the **specific task** to execute by **queering a central Database** server
- The **same Job** can execute **more than one task**
- The DB server provides also a **monitoring of the tasks execution**. It stores information such as:
 - Number of **failed executions** for each task
 - After a certain number of failures, the task should not be resubmitted again
 - **Job provenance** (which job has executed which task)
 - The exact **date** of the task execution
- It can manage **multiple task dependency**:
 - a task can be assigned to a Job only if all the tasks from which it depends were already completed successfully
 - Can be useful in running **workflows**



Generic “JST”: Submission procedure

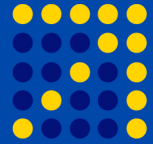
BioinfoGRID

- Based on **several scripts** which **runs automatically** at fixed time interval.
 - The **job submission** is made by some script **running as a daemon**
 - There is the possibility to **run more instances of the submission daemon** in order to **increase the total number** of job submitted in one hour
 - The **multi-process** submission **improve the speed** of submission
 - The submission uses **many RB** in a round robin algorithm in order **to avoid the over-load** of a single RB and to avoid that the **failure** of a single RB can stop the submission of jobs
- A different script retrieves periodically the OutputSandbox of the jobs
- Further **simple interactive scripts** are **provided to monitor** the status of the production by simply querying the monitoring DB
 - The user can know the **number of processed/running tasks**
 - The **number** of the **running** jobs
 - The **location** of each job
 - **Debug** eventual **errors** in running jobs
- The software to submit jobs is installed in more than one machine in order to avoid that a single hardware failure can stop the submission



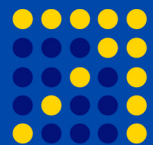
Generic “JST”: Running a task

- The **priority** of the task execution can be managed **dynamically**
 - It is an entry in the DB and can be used to choose next task to be assigned
- The **stage of the output** is managed in an advanced way
 - **Fail-over features** (more than one SE is used to avoid failures)
 - **Load-balancing features** (the SE is chosen randomly from a list of them)
- It is possible to send **monitoring information** from WN to a DB Server
 - It is useful to understand in **real time** what each job is doing
 - It is possible to send **whatever information** is needed
- The **scalability** can be increased **as needed**
 - **More servers** can be used to split the entire list of tasks in a round-robin configuration
 - However tests have shown that a single server can scale up to **10-20 thousand of concurrent jobs** (in case of tasks which take approximately 1 hour to be executed)
- Any **single point of failure** can be easily **avoided**
 - By using **more** the one RB, UI and SE
 - By **replicating** the “task-queue” DB server (tests have been performed with MySQL Replication option) **or splitting the task queue over many servers.**
- If there are some “**sleeping-jobs**” on WN’s this procedure can be used to run quickly **very short jobs with high priority**



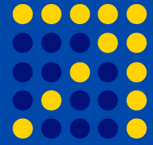
Generic “JST”: Component

- In order to submit and control automatically (without human control) all the tasks, we use a procedure made by few components:
 - DB Server -- Repository of all tasks
 - It is really useful to monitor the running job
 - It always have knowledge of done, running, undone and failed tasks
 - Can easily deal about prioritization, dependency, different kind of application or different users, etc.
 - Takes care of association between: task, executable, input and output files
 - In a separate table/DB there are also monitoring information like:
 - The exit status of each “error prone” action
 - CE/HOST of the execution
 - JOB_ID, TASK
 - STDOUT, STDERR
 - ...



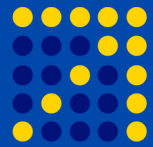
Generic “JST”: Component (2)

- In order to submit and control automatically (without human control) all the tasks, we use a procedure made by few component: (cont.)
 - Job Wrapper -- manage all the general action and make starts the real application
 - It takes care of “choosing” the task to be executed (querying the DB server)
 - It takes care of retrieving input files; storing output files (handling error conditions), updating the DB server accordingly
 - It launches the execution of the task and handles the exit_status updating the DB server accordingly
 - Job Submitter -- Submit job to different RB
 - It submits at different rates choosing the number of submitting process
 - It uses different RB in order to avoid single point of failure.
 - Job Controller -- Checks for task status and can choose action to be executed
 - It can retrieve automatically the output
 - Advice the user for the end of a task

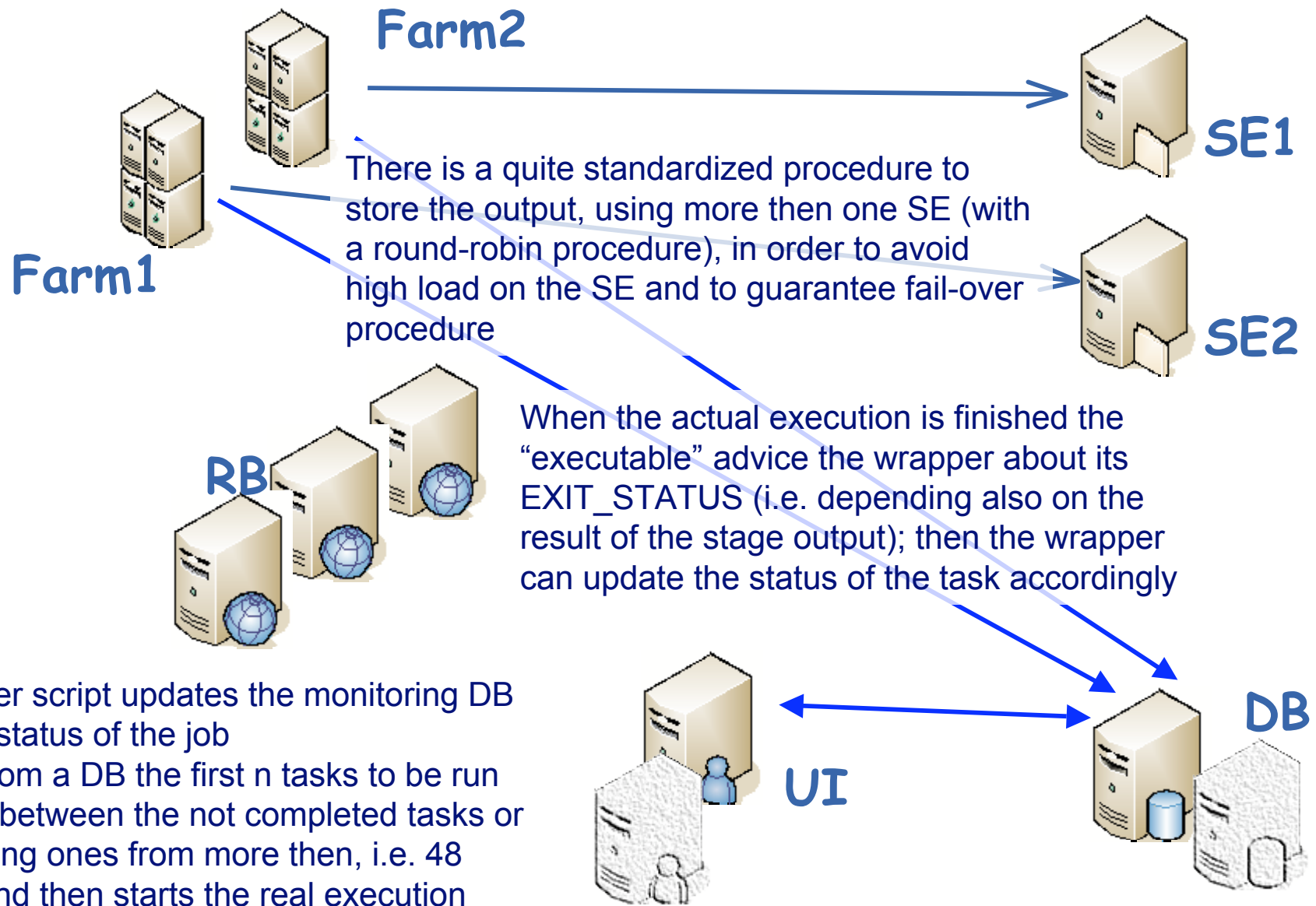


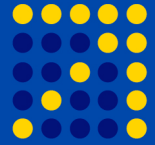
Generic “JST”: Use cases

- This procedure is also implemented in genius portal, to achieve:
 - An improved reliability
 - More control on the status and on the submission of jobs and tasks
 - The possibility to trigger action on changing task status
- The same procedure can be easily used to handle complex workflow
 - This should be split in elementary (atomic) tasks with the indication of each dependencies
- Or can be used to port new application:
 - It is needed to describe only the input/output files and the executable command line

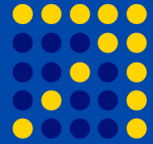


Generic "JST": Actions performed from a job in a WN

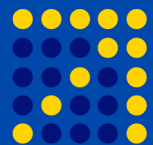




- Many challenges done in Bioinfogrid and LIBI project have been executed using this submission tool with fairly good performances and with a modest “human effort”.
- We are in contact with other researchers that are evaluating the tool:
 - DKFZ department: for running bioinformatics application on Grid
 - Pamela (Astrophysics experiment): in order to manage their data analysis on Grid



- **Bioinformatics applications:**
 - The **biggest problem** in running bioinformatics application is **data access**
 - Flat-file DB
 - RDBMS
 - **Problems dealing with flat-files:**
 - Usually the application is written **supposing "local" data access**
 - Often it is **not simple to modify** the code or it is not available
 - Some **WN** may **not** have **enough disk space** for the input/output files of some application
 - Often the solution of network **shared file-system is not practicable**
 - There can be problems of performances or local configuration
 - **For Accessing RDBMS ... please see next days**



How to access data

- Our goal is to **use the bio-applications as they are**, “wrapping” all file-system call
 - The user can “see” and access remote file with the same command used for local ones
- We have studied **two different software** that are actually capable to do that: **Parrot and FUSE**
 - The **first one** is most suitable for **batch execution**
 - The **second one** is most suitable for **interactive execution** (typically on a UI); since it uses a kernel module
- Both of them **support several protocols** (http, ftp, gsiftp)
 - But they do not exploit all the gLite DM functionality
- The overhead of the network access is a factor of 2 or less (compared with local disk access)

Local Access

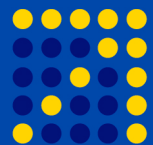
```
# cat myfile  
This is my file  
#
```

Access with FUSE

```
# httpfs http://myserver.ba.infn.it/ /fuse/  
# cat /fuse/myfile  
This is my file  
#
```

Access with Parrot

```
# parrot cat /http/myserver.ba.infn.it/myfile  
This is my file  
#
```



Why we need GFAL

- With **GFAL** it is possible to **exploit** all the functionalities of the **gLite DM**:
 - Use the **Logical File Name** (LFN) to hide the physical location of the files
 - And the complexity of the Physical File Name
 - **Hide** the different implementations of the **different SE's**
 - **Avoid** the **installation** of specific **software** to provide file access
 - Exploit the VOMS authentication
 - Exploit the Access Control List
 - Reducing the amount of byte transferred (only byte needed from application are really transferred)

```
# parrot ls -l /gfal-lfn/lfn/bio/myhome/
```

```
-rw-r--r-- 1 donvito donvito 2752 Mar 30 17:39 my_out_file  
-rw-r--r-- 1 donvito donvito 1244 Mar 30 17:39 my_input_file  
-rw-r--r-- 1 donvito donvito 1188 Mar 30 17:39 my_out_file2
```



BARI SE
dCache

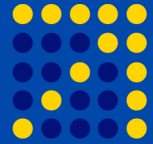


CNAF SE
CASTOR



LEGNARO SE
DPM





Parrot: some examples

1) Read, LFN

```
# echo "Hello World!!" > /home/donvito/test-parrot
```

```
# lgc-cr -l /grid/bio/donvito/myfiles/test-parrot file:///home/donvito/test-parrot
```

```
# parrot /bin/cat /gfal-lfn/grid/bio/donvito/myfiles/test-parrot
```

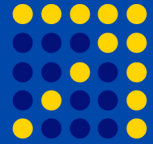
Hello World!!

2) Write, TURL

```
# parrot echo "Write Test" > /gsiftp/gridba6.ba.infn.it/flatfiles/SE00/bio/donvito/my-output-files/test-parrot-write
```

```
# parrot /bin/cat /gsiftp/gridba6.ba.infn.it/flatfiles/SE00/bio/donvito/my-output-files/test-parrot-write
```

Write Test



BioinfoGRID

Parrot: some examples (2)

3) Mountfile

```
# cat my-mount-file
```

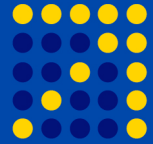
```
/test/parrot/file1 /gfa1-lfn/grid/bio/donvito/myfiles/test-parrot
```

```
/test/parrot/file2 /gsiftp/gridba6.ba.infn.it/flatfiles/SE00/bio/donvito/my-output-files/test-  
parrot-write
```

```
# parrot -m my-mount-file cat /test/parrot/file1 /test/parrot/file2
```

```
Hello World!!
```

```
Write Test
```



3) BLAST:

```
# export BLASTDB=/grid/bio/dkfz.de/db/ensembl_human/blast2
```

```
# cat dbtest/mountlist_ensembl_human
```

```
/grid/bio/dkfz.de/db/ensembl_human/blast2/ensembl_human_chromosome.22.nhr
```

```
/gsiftp/gridba6.ba.infn.it/flatfiles/SE00/bio/generated/2006-10-17/file76118e64-155c-4a19-ba15-6958c80bc40f
```

```
/grid/bio/dkfz.de/db/ensembl_human/blast2/ensembl_human_chromosome.22.nin
```

```
/gsiftp/gridba6.ba.infn.it/flatfiles/SE00/bio/generated/2006-10-17/filefebbc1c7-0fa7-4e0c-9a16-ee9e1c5ec2b1
```

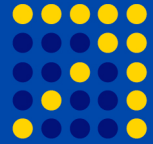
```
/grid/bio/dkfz.de/db/ensembl_human/blast2/ensembl_human_chromosome.22.nsd
```

```
/gsiftp/gridba6.ba.infn.it/flatfiles/SE00/bio/generated/2006-10-17/file05142c86-f390-4c63-86ea-e3200699ddab
```

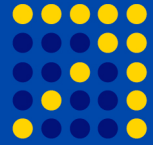
```
/grid/bio/dkfz.de/db/ensembl_human/blast2/ensembl_human_chromosome.22.nsi
```

```
/gsiftp/gridba6.ba.infn.it/flatfiles/SE00/bio/generated/2006-10-17/fileb0b431cb-d4d0-4be8-806e-ad4facda4375
```

```
# parrot -m dbtest/mountlist_ensembl_human blast-2.2.14/bin/blastall -p  
blastn -d ensembl_human_chromosome.22 -i sample-dna.seq.fa -o  
outfile.blastn2
```



- **Job Submission Tool:**
 - Used with success in many challenges run both over INFN-GRID and EGEE infrastructure
 - It makes the porting of new application really easy and reliable
 - It reduces the human effort for running huge and medium challenge
 - It can be easily added as to an existing portal in order to assure a greater reliability and increase monitoring capability
- **Parrot:**
 - It was useful with many applications tested in this period for BioinfoGrid and LIBI projects
 - Actually used for creating index and running BLAST
 - A group from DKFZ is using successful Parrot to run their bioinformatics applications
 - Still some work to do in order to address the problem of the two different RFIO implementations
 - The GFAL module is now in beta in the Parrot code, it is mainly based on the same procedure that we use in our implementation



- Parrot:
 - <http://webcms.ba.infn.it/cms-software/index.html/index.php/Main/GfalParrot>
- JST:
 - <http://webcms.ba.infn.it/cms-software/index.html/index.php/Main/JobSubmissionTool>
- For further information:
 - giacinto.donvito@ba.infn.it