KARLSRUHE INSTITUTE OF TECHNOLOGY

(KIT)

Shell Scripts for Corsika Code Simulation

(User Guide)

(Version 0.4 June 15,2011)

Himani Singla, Shenan Kalra and Sushant Sharma
Internship Students at SCC under the guidance of Dr. Gevorg Poghosyan

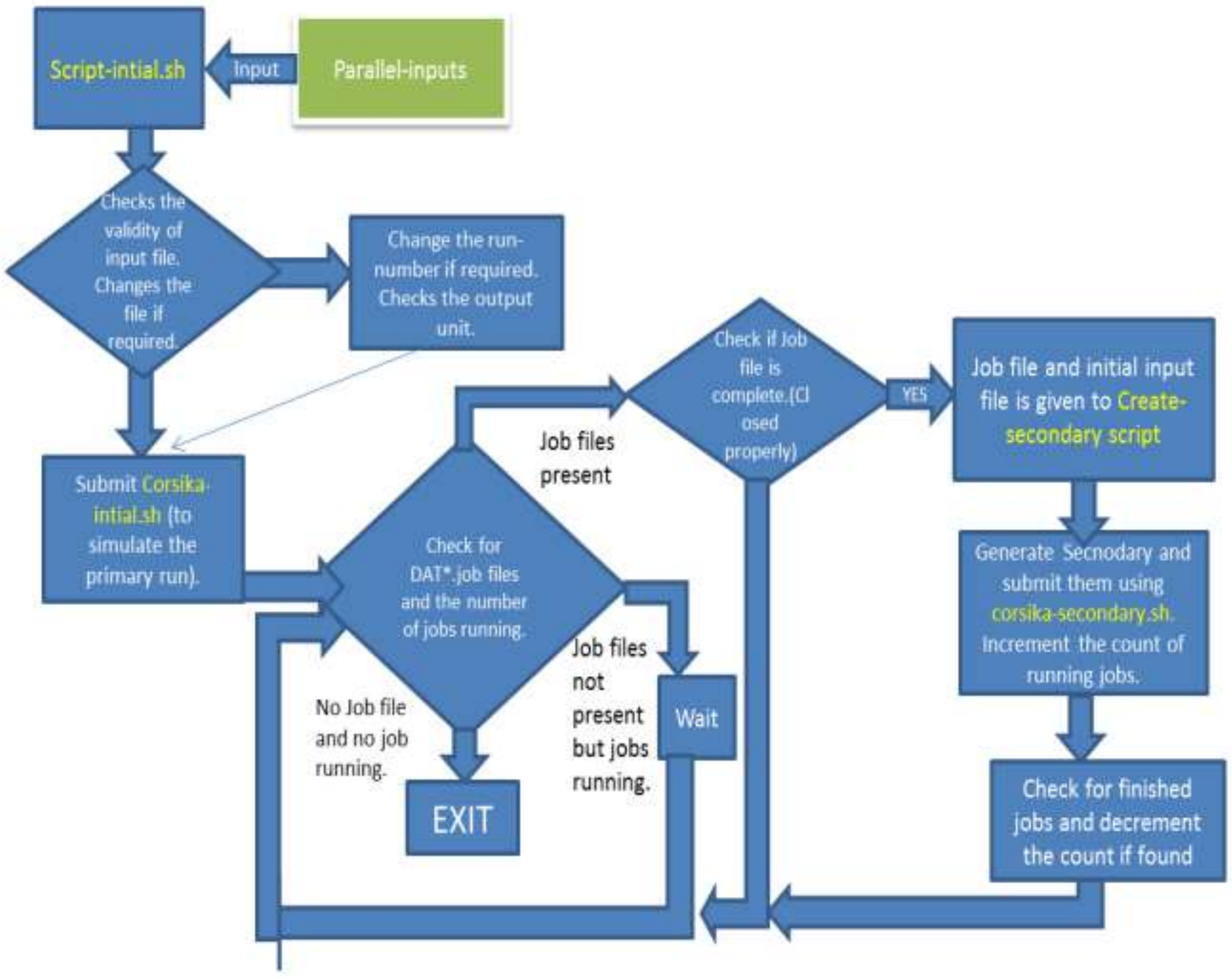
Abstract: We have developed Scripts for parallelized run of CORSIKA code. CORSIKA is a simulation code for extensive air showers initiated by high energy cosmic particles. The user guide explains the working of scripts and how they are used for running CORSIKA.

Detailed explanation:

The five scripts are:

1. corsika-initial.sh
2. scriptrunnr.sh
3. corsika-initial.ll.sh
4. create-secondary.sh
5. corsika-secondary.ll.sh

Flow Chart "Workflow of CORSIKA SCRIPTS"



Script-intial.sh
Input
Parallel-inputs
Checks the validity of input file. Changes the file if required.
Change the run-number if required. Checks the output unit.
Submit Corsika-intial.sh (to simulate the primary run).
Check for DAT*.job files and the number of jobs running.
Job files present
Check if Job file is complete.(Closed properly)
YES
Job file and initial input file is given to Create-secondary script
Generate Secnodary and submit them using corsika-secondary.sh. Increment the count of running jobs.
Check for finished jobs and decrement the count if found
Job files not present but jobs running.
Wait
No Job file and no job running.
EXIT

Working of Scripts:

1.1 Script-initial.sh

Script-initial.sh is an executable which runs locally. It is the script which is to be run initially. To run the script, issue the command:

./script-initial.sh parallel-inputs corsika-executable-name
where parallel inputs is the steering input file

It will automatically run the whole simulation.
Script-initial.sh forms a file called **level-file** which stores the information of job files, the level to which they belong and the number of DAT files they will form.
Script-initial.sh also forms **check-file** and **job-file** which are used by corsika-seconday.ll.sh to fill the required information in them (as explained in **corsika-seconday.ll.sh** section).

1.2 Scriptrunnr.sh

The script-initial.sh calls **scriptrunnr.sh** script which checks the run number of the input file and looks if any DAT file with that run number is present. If the file is present then it increments the run number of the input files by one.
It keeps on incrementing the run number until it finds any DAT file with same run number.

1.3 Corsika-initial.ll.sh

Script-initial.sh submits the input file using Loadleveler script called **corsika-initial.ll.sh**.
Corsika-initial.ll.sh script is based on the concept of multiple steps job in the file. In the first step the script updates the entry in the status_start status file and then runs the corsika executable with steering file as input and produces the output file. Then it waits until first job step is complete so that we can be sure that the run was successful and output and other files have appeared and are complete. In the second step it produces end file and writes the entry into status_finish status files.
After submitting the input file, we will get the following files:
Output files, DAT files, end file, output and error file of job called myjob.

end file is being used as a indicator to decrement the count of number of jobs running.

1.4 Create-secondary.sh

The script-initial.sh searches for the job files. After getting the any job file, script-initial.sh extracts the seed number and parent id of that job file. Then script-initial sends the initial input file, the job file and the seed number to the function called **function_start** inside the script called **create-secondary.sh**.
Then the **function_start** reads the job file sent to it line by line and forms **new input files** for further runs and appends initial input file inside the new file produced locally. Then it deletes the second instance of lines which are not required from the new input file produced. This input file is unique. So, from the name of the input file we can know from which job file it was made.

1.5 Corsika-secondary.ll.sh

Create-secondary.sh script submits the new input file to the Loadleveler script **corsika-secondary.ll.sh** just after creating it. **Corsika-secondary.ll.sh** script also uses the same concept of multiple step jobs as in **corsika-initial.ll.sh.** In the first step it updates status_start and runs corsika executable and produces output file. Also it fills **job-file** which contains information about the starting time, real time and CPU time of all the jobs. In the second step it updates status_finish, produces **end files** and fills the **check file** with the Loadleveler job ids of all submitted jobs.

Create-secondary.sh reads the next line of job file and follows the same procedure as explained above.

Then it goes back to **script-initial.sh**. The job file just used is moved into the job sub-directory for future reference. The script-initial.sh checks for the newer job files. If there are any new job files then again it uses the **create-secondary.sh** script and follows the same procedure.
And if there is no new job file i.e. the number of new job files is zero and there is no job running then it exits.

In the end we have files namely: DAT files, input files, output files, level-file, check-file and job-file, status files.

2.1 INPUTS:

a. Input Data file:

a)The steering file to be read by the CORSIKA program. Please Refer the CORSIKA User Guide.

b. Individual input files:

a) The initial input file is read and is modified to generate an individual input file for every parallel job. These input files can be found in the input sub-directory.

3.1 OUPTUT/STATUS:

a) CORSIKA Output:
The output files are saved in the directory *(running_directory)/output/*

The path of the directory for DAT and CUT files is given as input in the INPUT STEERING FILE. Kindly refer the CORSIKA user guide.

NOTE: to know more about DAT files and CUT files, please refer the CORSIKA User guide.

b) Script Runner output

I. *status_start*
Saved in the directory *(running_directory)/status/*
Gives the list of the jobs .
FORMAT:

```
1  0 0  1  0  1  0 0  1308148278
2 1  0  1 000000006 2 1 7 1308148282
3 1  0  1 000000006 3 8 8 1308148282
4 1  0  1 000000006 4 9 9 1308148282
5 1  0  1 000000006 5 10 19 1308148282
6 1  0  1 000000006 6 20 20 1308148282
7 1  0  1 000000006 7 21 23 1308148282
8 1  0  1 000000006 8 24 37 1308148282
9 1  0  1 000000006 9 38 38 1308148282
10 1  0  1 000000006 10 39 39 1308148283
11 1  0  1 000000006 11 40 45 1308148283
```

Explaination:

The First 2 coloums defines the unique MPI ID of the job and the MPI ID of the parent job.

The 3rd Coloum tells whether the job was a primary shower (1) or a secondary shower (0)

The coloums 4 to 6 defines the Run-number, seed, and MPIID which in-turn defines the name of the CUTFILE to be read by the CORSIKA shower.

The coloum 7 and 8 gives the index of the particle in the CUTFILE.

Coloum 9 is a double value which is the start time of the Shower/Sub-shower.

II. *status_finish1*

the list of Jobs finished.

FORMAT:

```
1  0 1  1  0  1  0 0  1308148282
3 1  0  1 000000006 3 8 8 1308148286
9 1  0  1 000000006 9 38 38 1308148286
6 1  0  1 000000006 6 20 20 1308148287
4 1  0  1 000000006 4 9 9 1308148287
20 6  0  1 586395971 20 81 81 1308148292
11 1  0  1 000000006 11 40 45 1308148297
10 1  0  1 000000006 10 39 39 1308148298
2 1  0  1 000000006 2 1 7 1308148301
21 6  0  1 586395971 21 82 97 1308148302
7 1  0  1 000000006 7 21 23 1308148305
8 1  0  1 000000006 8 24 37 1308148305
```

Explanation:

The First 2 coloums defines the unique MPI ID of the job and the MPI ID of the parent job.

The 3rd Coloum tells whether the job was a primary shower (1) or a secondary shower (0)

The coloums 4 to 6 defines the Run-number, seed, and MPIID which in-turn defines the name of the CUTFILE to be read by the CORSIKA shower.

The coloum 7 and 8 gives the index of the particle in the CUTFILE.

Coloum 9 is a double value which is the end time of the Shower/Sub-shower.

<u>2.1 Run TYPES:</u>

<u>By specifying the type of run you can select the architecture on which the simulation runs.</u>

Currently we support 2 types of runs.

1. LL -Load levellor.
   a. To run it use :

      ```
      ./script-initial.sh  parallel-inputs corsika-executable-name LL
      ```

      The simulation will run on distributed architecture which has LL as a scheduler.
2. Default:
   a. To run it use :

      ```
      ./script-initial.sh  parallel-inputs corsika-executable-name
      ```

      The simulation will run on the same system on which the scripts are executed. It can send the jobs in the background and can therefore benefit from the multicores in the same system.

3.1 Steps to run the scripts:

1. **After the compilation, copy the scripts from the 'parallel-sccripts' directory to the run directory.**
2. **Make sure that the run directory doesn't have job-files from any previous runs. If the script detects any old files, it will prompt you.**
3. **The parallel input files should not have the OUTPUT unit set to 89, which is for the MPI Runner version. If you are not sure, please remove the OUTPUT option from the input file.**
4. **The parallel input file should always contain the line:**

   ```
   PARALLEL T 1.000000000E+05 1.000000000E+06 1
   ```

   As the 1st line.
   Here the 2nd value should always be T (true) to treat the simulation parallel.
   The 3rd and the 4th values are the DECTCUT and DECTMAX respectively.
   The last value is the run-number (CAN BE CHANGED BY THE SCRIPT ITSELF IF NOT VALID)

5. **All the scripts should have the access privilege set to executable.**
6. **To run the script use :**

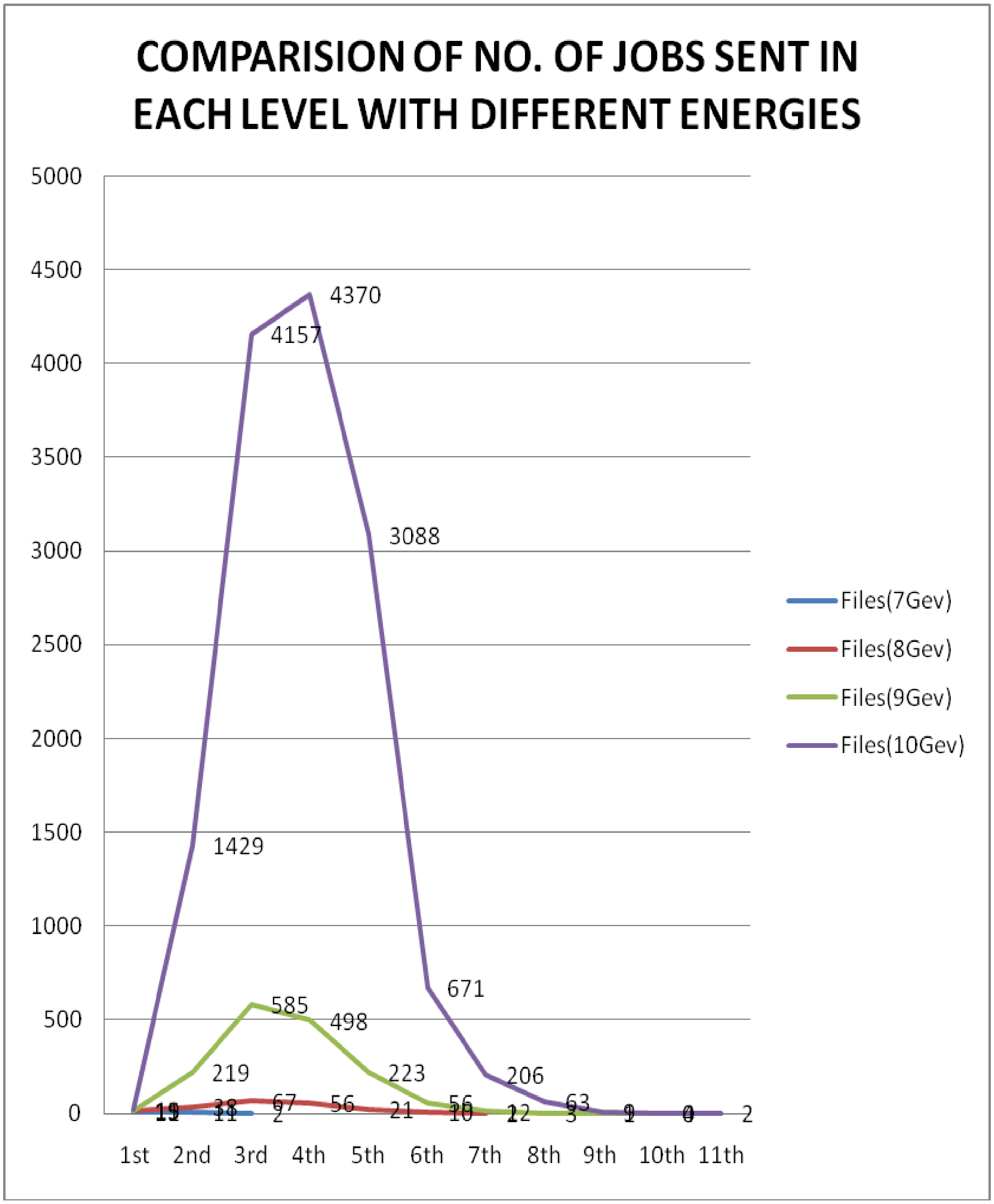
   ./script-initial.sh parallel-inputs corsika-executable-name ***type***
7. **The script will itself create some sub-directories. Please make sure you have the rights to do that.**
8. **Output files can be found in the *output* directory. The status files in the *status* directory. In the *job* directory you can find the used job files. The *end* directory should be empty after a successful run. *Input* directory contains the input files used by all the individual tasks.**

4.1 Some Observed Results:

The following table shows the number of job files produced in each step of each run with different energies in Gev. It also includes approximate time taken in minutes for complete run for different energies.

Energy Range (in Gev)				Number	of	jobs	produced					Total Jobs	Time taken (in mins)
	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	11th		
10^7	11	11	2									24	2 mins
10^8	15	38	67	56	21	10	2					209	19 mins
10^9	15	219	585	498	223	56	12	3	1	0		1612	150 mins
10^10	19	1429	4157	4370	3088	671	206	63	9	4	2	14008	1580 mins

The following graph shows the comparision between number of jobs produced and sent in each step for different energies:


COMPARISION OF NO. OF JOBS SENT IN EACH LEVEL WITH DIFFERENT ENERGIES
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
4370
4157
3088
1429
671
585
498
219
223
206
67
56
63
21
1st
2nd
3rd
4th
5th
6th
7th
8th
9th
10th
11th
Files(7Gev)
Files(8Gev)
Files(9Gev)
Files(10Gev)