SLURM Exercises Solutions

Eloi Vilella

2024-05-25

Introduction

This document contains the solutions to the SLURM exercises provided. Each exercise is rigorously solved with detailed explanations and justifications.

Exercise 1: Convert Text Files to Uppercase

Problem Statement

We have several text files (`names_1.txt`, `names_2.txt`, . . .). For each file, we want to generate a new file in uppercase letters (with a different extension). The `tr` command can be used for this action.

SLURM Job Script

```bash
#!/bin/bash
#SBATCH --job-name=uppercase_conversion
#SBATCH --output=uppercase_conversion_%j.out
#SBATCH --error=uppercase_conversion_%j.err
#SBATCH --ntasks=1
#SBATCH --time=00:10:00

for file in names_*.txt; do
  newfile="${file%.txt}.upper"
  tr '[:lower:]' '[:upper:]' < "$file" > "$newfile"
done
```

The script uses a simple loop to process each `names_*.txt` file. The `tr` command translates lowercase letters to uppercase and saves the result to a new file with the `.upper` extension. The SLURM directives specify the job name, output, error files, number of tasks, and time limit.

Exercise 2: Workflow Automation

Problem Statement

Write a script (either in bash or Python) that submits a workflow to: 1. Generate uppercase versions of each `names_*.txt` file. 2. Sort all files (original and uppercase) alphabetically. 3. Generate a summary file with the first and last name of each file using `summary.sh`.

Bash Script

```
#!/bin/bash

#Convert to uppercase
for file in names_*.txt; do
  sbatch --wrap="tr '[:lower:]' '[:upper:]' < $file > ${file%.txt}.upper"
done
wait

#Sort files
for file in names_*.txt ${file%.txt}.upper; do
  sbatch --wrap="sort $file -o $file"
done
wait

#Generate summary file
sbatch summary.sh names_*.txt ${file%.txt}.upper
```

Python Script

```
import os
import subprocess

#Convert to uppercase
files = [f for f in os.listdir() if f.startswith('names_') and f.endswith('.txt')]
for file in files:
    subprocess.run(['sbatch', '--wrap', f"tr '[:lower:]' '[:upper:]' < {file} > {file[:-4]}.upper"])
subprocess.run(['wait'])

#Sort files
all_files = files + [f"{file[:-4]}.upper" for file in files]
for file in all_files:
    subprocess.run(['sbatch', '--wrap', f"sort {file} -o {file}"])
subprocess.run(['wait'])

#Generate summary file
subprocess.run(['sbatch', 'summary.sh'] + all_files)
```

The scripts automate the workflow by submitting SLURM jobs for each task. They ensure that all steps are completed sequentially, leveraging SLURM's job scheduling capabilities. The `wait` commands ensure that each step is completed before moving on to the next.

Handling Job Dependencies

SLURM Job IDs and Dependencies

To handle job dependencies without using the `--parsable` option, we can capture the job ID and use it in subsequent job submissions.

Example Bash Script

```bash
#!/bin/bash

#convert to uppercase
job_ids=()
for file in names_*.txt; do
  job_id=$(sbatch --wrap="tr '[:lower:]' '[:upper:]' < $file > ${file%.txt}.upper" | cut -d ' ' -f 4)
  job_ids+=($job_id)
done

#wait
dependency=$(IFS=:; echo "${job_ids[*]}")

#sort files
job_ids=()
for file in names_*.txt ${file%.txt}.upper; do
  job_id=$(sbatch --dependency=afterok:$dependency --wrap="sort $file -o $file" | cut -d ' ' -f 4)
  job_ids+=($job_id)
done

#wait
dependency=$(IFS=:; echo "${job_ids[*]}")

#generate summary file
sbatch --dependency=afterok:$dependency summary.sh names_*.txt ${file%.txt}.upper
```