PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN

IIC2523 Sistemas Distribuidos (II/2017)

Actividad 2

Raimundo Herrera (rjherrera@uc.cl)

1 Local v/s Slurm

Al usar de manera local uno de los nodos disponibles, el rendimiento es mucho mejor que al usar *slurm*, esto puede deberse a que la tarea es muy sencilla, se discutirá más en profundidad más adelante.

Para probar lo anterior, hice un *script* secuencial de python, el cual es idéntico al que se ejecuta con *sbatch*, pero en vez de recibir como parámetro el `TASK_ID` y usarlo para definir el intervalo, simplemente utilicé el intervalo completo, en mi caso de 0 a 100100.
Al utilizar el comando

```
time python3 primes_seq.py
```

Es decir, cronometrar la ejecución secuencial del programa, en `titan`, se obtiene un tiempo de ejecución de 0.359 segundos promedio de 5 ejecuciones.

Por otro lado, al realizar `sbatch` con intervalos de largo 100 y 1000 ejecuciones, se obtiene el resultado tras 60 segundos. Esto se obtuvo utilizando el comando

```
scontrol show job <job_id>
```

y revisando los tiempos de inicio y término de cada *job*, para así, obteniendo el menor tiempo de inicio y el mayor tiempo de finalización, obtener el tiempo total.

La diferencia podría darse por varias razones, al ser trabajos medianamente cortos, e intervalos pequeños, el tiempo de trabajo que toma para cada nodo, es muy corto, por lo que existe un mayor delta de tiempo al ir distribuyendo las tareas, que en la ejecución misma, dicho de otra manera, hay un overhead en la distribución y no en la ejecución.

Por otro lado, también puede explicarse porque se están realizando múltiples operaciones de `I/O` que, por la misma razón que antes, ya que el computo es muy sencillo, pueden resultar más costosas que la ejecución misma.

2 Nodos fijos

Para asignar una cantidad fija de nodos en el *script*, lo que se podría hacer es agregar el *flag* `--nodes` disponible para el comando `sbatch`. Este comando, como se muestra en la documentación, sirve para determinar la cantidad mínima, y opcionalmente máxima, de nodos a utilizar para un trabajo, de modo que al setear ambos, se puede obtener el comportamiento deseado.

3 Distribución Slurm

Slurm se basa en el algoritmo de *Backfill scheduling*. Este *scheduler* solo comienza los trabajos de menor prioridad si al hacerlo no empeora el desempeño de los de mayor prioridad.

El *workflow* general, es de la siguiente manera

1. Un *job* se añade a la cola
2. Se asigna una prioridad al *job*: Esta se asigna por medio de:
    - Fairshare: Prioriza trabajos de quienes hayan tenido menos trabajos anteriormente (*underservices*). Evita *starvation*.
    - Age: Aumenta la prioridad a medida que el trabajo ha pasado más tiempo, es decir también evita *starvation*.
    - Tamaño (*Job Size*): Se favorece a los trabajos que requieran mayor cantidad de CPUs.
3. El *job* espera en cola hasta que le toque, es decir, los recursos esten disponibles y no haya otros de mayor prioridad que el

De esta manera, al asignar prioridad por medio de esas tres medidas, se van asignando los trabajos en orden prioritario.

Respuesta basada en ACCRE Intro to SLURM