GNU Parallel 조사 3 - 찬혁

9. Pipe mode

- GNU Parallel은 명령어 템플릿에 값을 넣는 대신, 표준 입력(stdin)을 명령어로 파이프를 통해 전달할 수 있다.
- `--pipe` 기능은 GNU Parallel을 다른 모드로 전환한다. 즉, 표준 입력 데이터를 명령어 실행을 위한 인수로 처리하는 대신, 데이터를 명령어의 표준 입력(stdin)으로 전달한다.
- 일반적인 상황은 다음과 같다.

```shell
command_A ¦ command_B ¦ command_C
```

- 여기서 `command_B`가 느릴 경우, 이를 병렬로 실행하여 속도를 높이고자 한다.
    - 3장에서 제공된 테스트 파일이 필요

9.1 Block size

- 기본적으로 GNU Parallel은 `command_B`의 인스턴스를 시작하고, 1MB의 블록을 읽은 다음, 가장 가까운 레코드를 찾아 해당 청크(chunk)를 해당 인스턴스에 전달한다.
- 그런 다음 다른 인스턴스를 시작하고, 또 다른 블록을 읽어 가장 가까운 레코드를 찾아 두 번째 인스턴스에 해당 청크를 전달한다.
- 예제

```shell
cat numi000000 ¦ parallel --pipe wc
```

- 출력 (순서는 다를 수 있음)

```shell
165668 165668 1048571
149797 149797 1048579
149796 149796 1048572
149797 149797 1048579
149797 149797 1048579
149796 149796 1048572
85349 85349 597444
```

- 블록의 크기가 정확히 1MB가 아닌 이유는 GNU Parallel이 절대 줄의 일부만 전달하지 않고 항상 전체 줄을 전달하기 때문이다.
- 따라서 블록 크기는 평균적으로 1MB 이다.
- `--block` 옵션을 사용하여 블록 크기를 2MB로 변경할 수 있다.

```shell
cat num1000000 ¦ parallel --pipe --block 2M wc
```

- 출력(순서는 다를 수 있음):

```shell
315465 315465 2097150
299593 299593 2097151
299593 299593 2097151
 85349  85349  597444
```

- GNU Parallel은 각 줄을 하나의 레코드로 처리한다.
- 레코드의 순서가 중요하지 않은 경우(예: 모든 줄을 처리해야 하지만 어떤 줄이 먼저 처리되는지는 중요하지 않은 경우) `--round-robin` 옵션을 사용할 수 있다.
- `--round-robin` 옵션이 없을 경우, GNU Parallel은 블록당 하나의 명령을 실행한다.
  - 하지만 `--round-robin`을 사용하면 요청된 작업 수(-j 또는 --jobs)에 따라 실행된다.
- 레코드는 실행 중인 작업 사이에 분배된다.

```shell
cat numi0e@0000 ¦ parallel --pipe -j4 --round-robin wc
```

- 출력 예:

```shell
149797 149797 1048579
299593 299593 2097151
315465 315465 2097150
235145 235145 1646016
```

- 4개의 인스턴스 중 1개는 단일 청크를, 2개는 각각 2개의 전체 청크를, 1개는 1개의 전체 청크와 1개의 일부 청크를 받는다.
- `--round-robin`은 준비가 된 첫 번째 프로세스에 청크를 전달한다.
- 청크의 순서를 각 프로세스에 엄격히 지정하려면 `--keep-order` 옵션을 사용하면 된다.

```shell
cat num1000000 ¦ parallel --pipe -j4 --keep-order --round-robin wc
```

- 출력

```shell
315464 315464 2097143
299592 299592 2097144
235148 235148 1646037
149796 149796 1048572
```

9.2 Records

- GNU Parallel은 입력을 **레코드**로 인식한다.
  - 기본(default) 레코드는 한 줄이다.
- `-N140000` 옵션을 사용하면 GNU Parallel은 한 번에 140,000개의 레코드를 읽는다.
- 예시:

```shell
cat num1000000 ¦ parallel --pipe -N140000 wc
```

- 출력 (순서는 다를 수 있음):

```shell
140000 140000 868895
140000 140000 980000
140000 140000 980000
140000 140000 980000
140000 140000 980000
140000 140000 980000
140000 140000 980000
20000  20000  140001
```

- 마지막 작업은 전체 140,000줄을 처리하지 못하고, 대신 20,000줄만 처리한 것을 알 수 있다.
- 한 레코드가 75줄로 구성된 경우에는 `-L` 옵션을 사용할 수 있다.
- 예시:

```shell
cat numi000000 | parallel --pipe -L75 wc
```

- 출력 (순서는 다를 수 있음):

```shell
165600 165600 1048095
149850 149850 1048950
149775 149775 1048425
149775 149775 1048425
149650 149850 1048950
149775 149775 1048425
65350  85350  597450
25     25     176
```

- GNU Parallel은 여전히 약 1MB 크기의 블록을 읽지만, `wc` 명령어에 전달할 때는 한 번에 75줄씩 보낸다.
- 물론, 마지막 작업은 예외이다.
    - 위의 경우, 마지막 작업은 25줄을 처리했다.

9.3 Record separators

- GNU Parallel은 레코드가 어디에서 나뉘는지를 결정하기 위해 **구분자(separators)** 를 사용한다.
- `--recstart`는 레코드가 시작되는 문자열을 지정하고, `--recend`는 레코드가 끝나는 문자열을 지정한다.
- 기본값은 `--recend '\n'` (줄바꿈)과 `--recstart ""` 이다.
- `--recend`와 `--recstart`를 모두 설정하면, 레코드는 `--recend` **문자열 바로 뒤에** `--recstart` **문자열이 따라올 때만** 나뉜다.

예제 1: 입력 데이터 분리하기

- 다음 입력 데이터를 사용한다고 가정한다.

```shell
/foo, bar/, /baz, qux/,
```

- 이 데이터를 다음과 같이 분리하고 싶다.

```shell
/foo, bar/,
/baz, qux/,
```

---

1. `--recend`만 사용한 경우

- `--recend`를 `', '`로 설정:

```shell
echo /foo, bar/, /baz, qux/, ¦ \
        parallel -kN1 --recend ', ' --pipe echo JOB{@}\;cat\;echo END
```

- 출력

```shell
JOB1
/foo, END
JOB2
bar/, END
JOB3
/baz, END
JOB4
qux/,
END
```

- 원하는 결과가 아님.
  - 레코드에 `', '`가 포함되어 있기 때문

---

2. `--recstart`만 사용한 경우

- `--recstart`를 `'/'`로 설정

```shell
echo /foo, bar/, /baz, qux/, ¦ \
        parallel -kN1 --recstart / --pipe echo JOB{@}\;cat\;echo END
```

- 출력

```
JOB1
/foo, barEND
JOB2
/, END
JOB3
/baz, quxEND
JOB4
/,
END
```

- 이것도 원하는 결과가 아님.

---

3. `--recend`와 `--recstart`를 모두 설정한 경우

- `--recend`를 `', '`로, `--recstart`를 `'/'`로 설정:

```
echo /foo, bar/, /baz, qux/, | \
        parallel -kN1 --recend ', ' --recstart / --pipe \
        echo JOB{@}\;cat\;echo END
```

- 출력

```
JOB1
/foo, bar/, END
JOB2
/baz, qux/,
END
```

- 원하는 결과가 나왔다.

정규식 사용하기

- `--regexp` 옵션을 사용하면 `--recend`와 `--recstart`를 정규식으로 처리할 수 있다.

```shell
echo foo,bar,_baz,__qux | \
        parallel -kN1 --regexp --recend ,_* --pipe \
        echo JOB{@}\;cat\;echo END
```

- 출력

```shell
JOB1
foo, END
JOB2
bar, END
JOB3
baz, __END
JOB4
qux
END
```

레코드 구분자 제거

- `--remove-rec-sep` 또는 `--rrs` 옵션을 사용하면 레코드 구분자를 제거할 수 있다.

```shell
echo foo,bar,_baz,__qux | \
        parallel -kN1 --rrs --regexp --recend ,_* --pipe \
        echo JOB{@}\;cat\;echo END
```

- 출력

```shell
JOB1
fooEND
JOB2
barEND
JOB3
bazEND
JOB4
qux
END
```

9.4 Header

- 입력 데이터에 **헤더**가 있는 경우, `--header` 옵션을 사용해 각 작업에 헤더를 반복적으로 추가할 수 있다.

예제 1: 정규 표현식으로 헤더 매칭하기

- 헤더가 `%`로 시작한다고 가정

```shell
cat num_%header | \
        parallel --header '(%.*\n)*' --pipe -N3 echo JOB{@}\;cat
```

- 출력 (순서는 다를 수 있음)

```shell
JOB1
%head1
%head2
1
2
3
JOB2
%head1
%head2
4
5
6
JOB3
%head1
%head2
7
8
9
JOB4
%head1
%head2
10
```

예제 2: 헤더가 2줄일 경우

- 헤더가 정확히 두 줄이라면 `--header 2`를 사용할 수 있다.

```shell
cat num_%header | parallel --header 2 --pipe -N3 echo JOB{@}\;cat
```

- 출력: 위와 동일

9.5 Fixed length records

- **고정 길이 레코드(Fixed length records)** 는 `--recend ''`와 `--block recordsize`를 설정해 처리할 수 있다.
- n 바이트 크기의 헤더는 `--header .{n}`를 사용해 처리한다.

예제: 4바이트 헤더와 3바이트 레코드

```shell
cat fixedlen | parallel --pipe --header .{4} --block 3 --recend '' \
        'echo start; cat; echo'
```

- 출력

```shell
start
HHHHAAA
start
HHHHCCC
start
HHHHBBB
```

9.6 Programs not reading from stdin

- 일부 프로그램은 **표준 입력(stdin)** 대신 파일로부터 데이터를 읽어야 한다.

9.6.1 `--cat` 사용

- `--cat` 옵션을 사용하면, GNU Parallel이 임시 파일을 생성해 데이터를 저장한 후 해당 파일을 프로그램이 읽도록 한다.
- 프로그램이 종료되면 임시 파일은 삭제된다.

```shell
cat num1000000 | parallel --pipe --cat wc {}
```

- 출력

```shell
149796 149796
165668 165668
149796 149796
...
```

- GNU Parallel은 `/tmp/parXXXXX` 형식의 임시 파일을 생성하고, 각 파일에 데이터를 저장한 뒤 `wc` 명령어를 실행한다.
- 프로그램이 끝나면 이 파일들은 삭제된다.

9.6.2 `--fifo` 사용

- `--cat`은 데이터를 디스크에 저장한 뒤 읽기 때문에 느릴 수 있다.
- 프로그램이 **FIFO**(명명된 파이프)를 읽을 수 있다면, GNU Parallel은 데이터를 디스크에 저장하지 않고 처리할 수 있다.

```shell
cat num1000000 | parallel --pipe --fifo wc {}
```

- 출력

```shell
149796 149796 1048572 /tmp/parXXXXXX
165668 165668 1048571 /tmp/parYYYYYY
...
```

- FIFO 제한사항
  - 프로그램이 파일을 **처음부터 끝까지** 읽어야만 한다.
  - 만약 일부 데이터만 읽고 처리를 중단한다면, GNU Parallel은 작업이 멈추게 된다.

9.7 Use `--pipepart` for high performance

- `--pipe`는 성능이 그리 효율적이지 않아 약 500MB/s의 속도에 그친다.
- 반면, `--pipepart`는 5GB/s 이상의 속도를 쉽게 낼 수 있다.
- 하지만 몇 가지 제한 사항이 있다.

1. 입력 데이터는 반드시 일반 파일이거나 블록 디바이스여야 한다.
(파이프나 FIFO는 사용할 수 없다.)

2. 입력은 `-a` 또는 `::::`로 지정해야 한다.
3. `-L`, `-1`, `-N` 옵션은 사용할 수 없다.
   - 그러나 `--recend`와 `--recstart`는 사용할 수 있어, 이를 기반으로 레코드를 나누는 것이 가능하다.

예제: `--pipepart`로 데이터 처리

- 다음 명령은 데이터를 3MB 단위로 나눠 처리한다.

```shell
parallel --pipepart -a num1000000 --block 3m wc
```

- 출력 (순서는 다를 수 있음)

```shell
444443 444444 3000002
428572 428572 3000004
126985 126984  888890
```

`--block`에 음수 값 지정하기

- `--block`에 음수 값을 설정하면 각 작업 슬롯이 처리해야 할 블록의 수를 의미한다.
- 예를 들어, 다음 명령은 3 * 5 = 15개의 작업을 실행합니다:

```shell
parallel --pipepart -a num1000000 --block -3 -j5 wc
```

효율적인 대안

- 이 방식은 `--round-robin`보다 효율적이다.
  - GNU Parallel이 데이터를 읽지 않고 직접 처리하기 때문
- 소수의 작업 슬롯으로도 대규모 데이터를 처리할 수 있다.

입력 순서를 유지하기: `--keep-order`

- `--keep-order` 를 사용하면 입력 데이터의 순서와 동일하게 출력할 수 있다.
- `--round-robin` 방식에서는 입력이 섞이기 때문에 순서를 유지할 수 없다.

9.8 Duplicate all input using `--tee`

- `--tee` 옵션을 사용하면 동일한 입력 데이터를 여러 작업에 복제할 수 있다.

예제:

- 다음 명령은 입력 데이터를 각 작업에 복제해 `grep` 명령어를 실행한다.

```shell
seq 30 | parallel -v --pipe --tee --tag grep {} ::: 4 5 6
```

- 출력

```shell
4 grep
4 4
4 14
4 24
5 grep 5
5 5
5 15
5 25
6 grep 6
6 6
6 16
6 26
```