PFN インターン選考課題 2019 バックエンド分野

福岡 拓也

実行環境

- macOS Mojave (10.14.4)
- Python 3.7.1
  - Flask 1.0.2
- Apple LLVM version 10.0.1 (clang-1001.0.46.4)

問題 1.1 (サーバーの作成)

サーバープログラムの開発には, Python の軽量フレームワーク Flask を用いた. 環境変数 DATA_DIR でジョブの存在するディレクトリを指定することができる. Created (作成日時) はタイムスタンプではなく秒に直したものを, Priority (優先度) については, High の場合は 1 を, Low の場合は 0 を返すような仕様とした. また, 同じ時刻に対して複数のジョブを返すことが可能となっている.

```
$ cd server
$ DATA_DIR=../test/data_test1 FLASK_APP=server.py flask run &
$ curl http://localhost:5000/jobs/00:00:03

{"Jobs":[{"Created":3,"JobID":2,"Priority":1,"Tasks":[3,5]}]}
```

サーバーについてのユニットテストは以下のコマンドで走らせることができる.

```
$ cd server
$ DATA_DIR=./data FLASK_APP=server.py flask run &
$ python test.py
```

問題 1.2 (ワーカーの作成)

ワーカーは以下のコマンドで走らせることができる. 標準出力には時刻と Executing Point を, エラー出力にはジョブが終了した時にそのジョブの優先度と, 待ち時間 (終了時間 - 開始時間 - タスクの総量) を出力する.

```
$ cd server
$ FLASK_APP=server.py flask run &
$ cd ../worker
$ make
$ MAXTIME=4000 CAPACITY=-1 PRIORITY_RANGE=1 ./worker 1>../plot/log1_2 2>../plot/wait1_2
```

サーバーとワーカーについての統合テストは以下のように走らせることができる. 統合テスト内でサーバーの起動と終了を行なっているので, 予めサーバーを起動させる必要はない.

```
$ cd test
$ ./test_run.sh
```

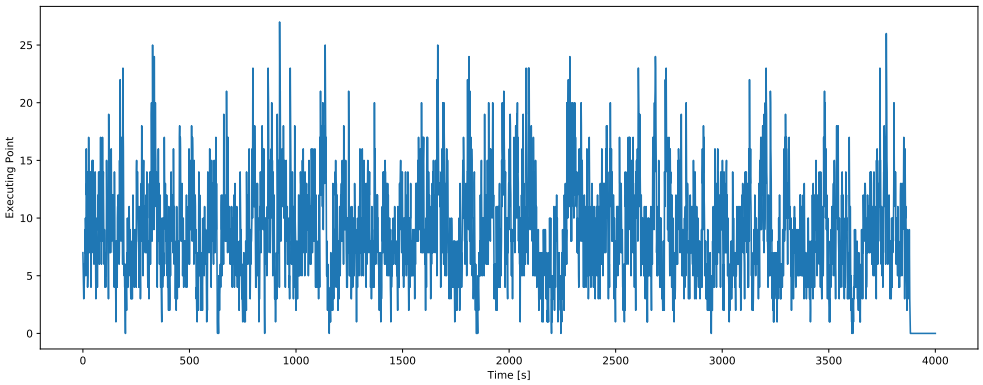

Figure1　The transition of Executing Point with no capacity and no priority (1-2)

問題 2.1 (キャパシティの追加)

実行していないジョブと実行されているジョブをそれぞれ別のリストで管理し, 実行時に移動させるような設計にしている.

```
$ cd server
$ FLASK_APP=server.py flask run &
$ cd ../worker
$ make
$ MAXTIME=4000 CAPACITY=15 PRIORITY_RANGE=1 NAIVE=1 ./worker 1>../plot/log2_1 2>../plot/wait2_1
```

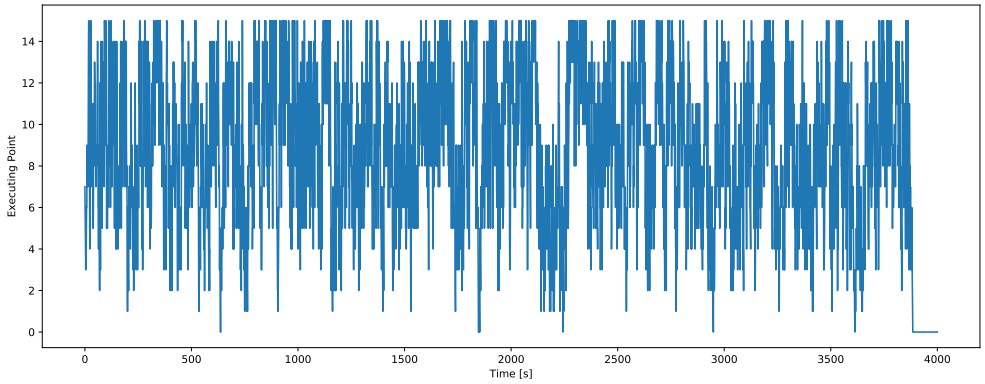

Figure2　The transition of Executing Point with no priority (2-1)

問題 2.2 (優先度の追加)

さらに, 優先度毎に別のリストでジョブを管理するようにした. また, 高い優先度のジョブの実行の妨げが生じないように, 実行を待っている高い優先度のジョブが存在する時には, 低い優先度のジョブの実行を一切行わない設計にしている. 例えば, Executing Point の余裕が 5 であり, 低い優先度のジョブの次に実行されるタスクの大きさが 3 で, 高い優先度のジョブの次に実行されるタスクの大きさが 6 の場合, 低い優先度のジョブの実行は見送ることになる. 実行時には問題 2.3 の実装と区別するために, 環境変数 NAIVE=1 を設定する.

```
$ cd server
$ FLASK_APP=server.py flask run &
$ cd ../worker
$ make
$ MAXTIME=4000 CAPACITY=15 PRIORITY_RANGE=2 NAIVE=1 ./worker 1>../plot/log2_2 2>../plot/wait2_2
```

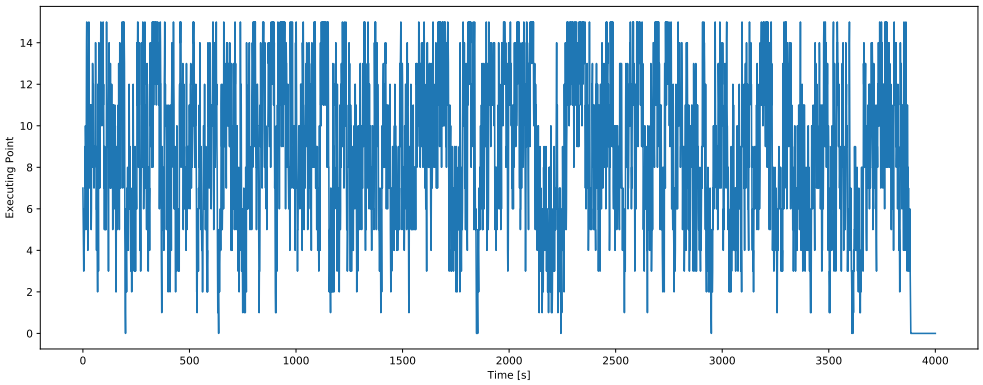

Figure3　The transition of Executing Point with the naive scheduling (2-2)

問題 2.3 (実行スケジューリングの効率化)

```
$ cd server
$ FLASK_APP=server.py flask run &
$ cd ../worker
$ make
$ MAXTIME=4000 CAPACITY=15 PRIORITY_RANGE=2 ./worker 1>../plot/log4 2>../plot/wait2_3
```

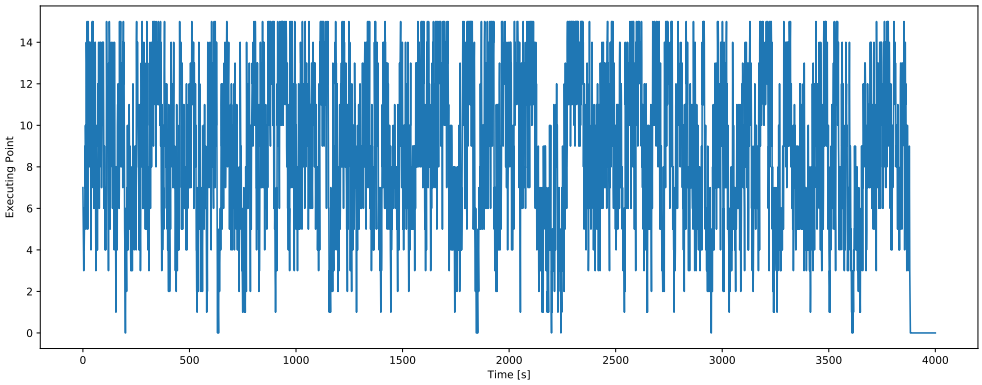

Figure4　The transition of Executing Point with the efficient scheduling (2-3)

問題 2.2 のジョブのスケジューリングについて, 高い優先度のジョブが存在する時に無条件で低い優先度のジョブの実行を見送るのは効率が悪い. したがって, ジョブのスケジューリングの先読みを行い, 近い将来行われる高い優先度のジョブの実行に妨げのない場合にのみ限り低い優先度のジョブの実行を許可するようにする. 先読みの秒数は長い方がより正確に予測ができるが計算量が大きくなるという欠点がある. そのトレードオフを考えて, 今回はキャパシティの大きさと同じだけの秒数だけの先読みを行うことにする. そうすることで, 少なくとも直近の高い優先度のジョブについては, 低い優先度のジョブの実行に妨げられず実行ができることが保証できる. 以下具体例を用いて説明する.

まず, 現在実行されているジョブの影響のみを考えた, Executing Point の余裕分 (以下 Spare Point と呼ぶ) の推移を計算する. キャパシティが 8 で, 2 と 4 の大きさのタスクが実行されていたとすると, Spare Point は

2, 4, 6, 7, 8, 8, 8, 8, 8

と推移する. また, この時優先度の高いタスクの大きさが 5 のジョブが実行を待っているとすると, このジョブは最短で 2 秒後に実行することができるとわかり, このジョブが実際 2 秒後に実行されるとすると, この時の Spare Point の推移は

2, 4, 1, 3, 5, 6, 7, 8, 8

と予測できる. 同様に, 優先度の高い次のジョブのタスクの大きさが 3 であった場合には, このジョブは 3 秒後に実行することができて, このジョブの実行を予約した時の Spare Point の推移は

2, 4, 1, 0, 3, 5, 7, 8, 8

となる. これを待機している優先度の高いジョブについて全て適応して, その後に初めて, それでも実行できる優先度の低いジョブを実行する. 補足であるが, 現在の Spare Point が 0 になった時点で探索を終了して次の時刻に進むようにすることで, 計算量を削減している.

待ち時間を, ジョブ終了時刻 - ジョブ生成時刻 - タスクの総量と定義して, 改善前と改善後の待ち時間をヒストグラムで比較したのが次のグラフである. 待ち時間の平均は, 改善前の実装については高い優先度と低い優先度のジョブそれぞれについて, 1.88 と 0.208 であるのに対して, 改善後については 0.81 と 0.0755 であり, 2 倍以上待ち時間が改善されていることがわかる.

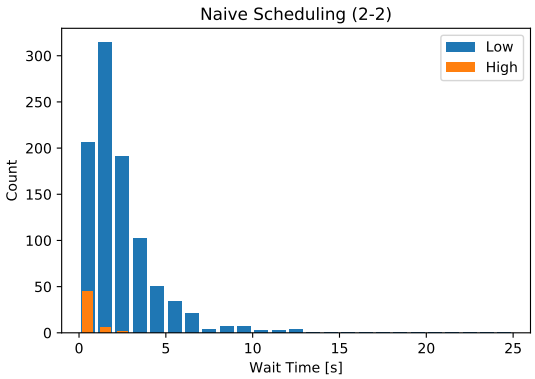

Figure5　Count of waiting time in naive implementation (2-2)

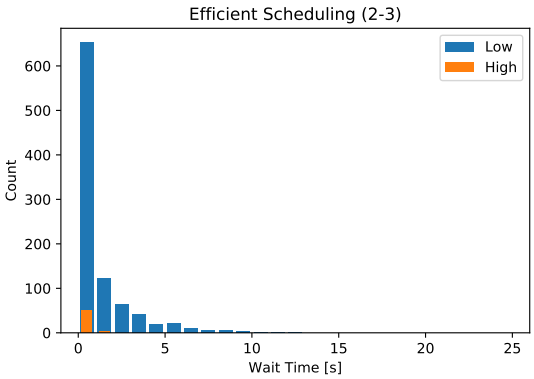

Figure6　Count of waiting time in efficient implementation (2-3)

問題 3.1 (0-100 までの優先度に対応)

```
$ cd server
$ DATA_DIR=./data_num_priority NUM_PRIORITY=1 FLASK_APP=server.py flask run &
$ cd ../worker
$ make
$ MAXTIME=4000 CAPACITY=15 PRIORITY_RANGE=101 ./worker 1>../plot/log3_1 2>../plot/wait3_1
```

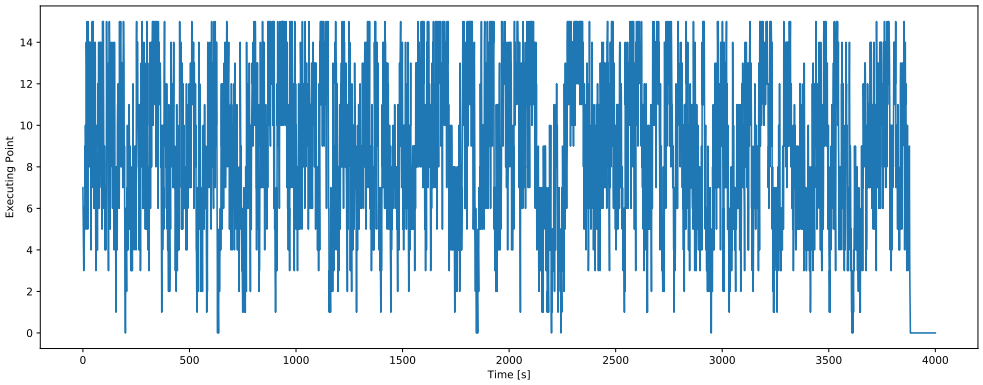

Figure7　The transition of Executing Point with the efficient scheduling (3-1)

問題 2-3 のアルゴリズムに関しては, 優先度の個数が増えた場合でも同様に適用することができる. 問題 2-1 から問題 2-2 の拡張と同じように, 優先度の個数だけジョブを管理するリストを用意すれば, それがそのまま問題 3-3 の実装となる.