ISUCON 勉強会 4章

前回の復習

- 負荷試験をソフトウェア的に行うツールとして、Apache Bench がある。
- ウェブアプリケーションのチューニングは、負荷試験中のリソースのモニタリングや、試験後のログの解析をすることによってボトルネックを特定し改善していくというプロセスによって行われる。
  - リソースのモニタリング: top, dstat など
  - nginx ログ解析: alp, kataribe など
  - mysql ログ解析: mysqldumpslow, pt-query-digest など

k6

- Apache Bench よりリッチな機能が備わる負荷試験ツール
- 特定のエンドポイントを叩くだけでなく、ホーム画面 -> ログイン -> 画像の投稿 -> フィードに行くというようなより実際に近い負荷試験を行うことができる。

環境準備

- Prerequisites:
  - docker
  - docker-compose
  - git
  - subscription to techresi-isucon-workshop

```
$ cd private-isu-enshu/webapp/sql

$ curl -L -O
https://github.com/catatsuy/private-isu/releases/download/img/dump.sql.bz2

$ cd .. && docker compose up
```

単純な負荷試験

```
$ k6 run --vus 1 --duration 30s ab.js
```

- --vus: number of virtual users
- --duration: time to run the experiment

出力

```
running (0m30.1s), 0/1 VUs, 21 complete and 0 interrupted iterations
default ✓ [======================================] 1 VUs  30s

     data_received..................: 678 kB 23 kB/s
     data_sent......................: 1.5 kB 50 B/s
     http_req_blocked...............: avg=84.8µs   min=2.52µs  med=4.06µs   max=1.62ms   p(90)=6.7µs    p(95)=81.53µs
     http_req_connecting............: avg=12.17µs  min=0s      med=0s       max=255.64µs p(90)=0s       p(95)=0s
     http_req_duration..............: avg=1.43s    min=1.03s   med=1.43s    max=1.55s    p(90)=1.5s     p(95)=1.53s
       { expected_response:true }...: avg=1.43s    min=1.03s   med=1.43s    max=1.55s    p(90)=1.5s     p(95)=1.53s
     http_req_failed................: 0.00%  ✓ 0        ✗ 21
     http_req_receiving.............: avg=475.68µs min=156.5µs med=467.39µs max=1.13ms   p(90)=981.43µs p(95)=1.1ms
     http_req_sending...............: avg=32.99µs  min=12.16µs med=26.62µs  max=76.64µs  p(90)=59.7µs   p(95)=68.62µs
     http_req_tls_handshaking.......: avg=0s       min=0s      med=0s       max=0s       p(90)=0s       p(95)=0s
     http_req_waiting...............: avg=1.43s    min=1.03s   med=1.43s    max=1.55s    p(90)=1.5s     p(95)=1.53s
     http_reqs......................: 21     0.698022/s
     iteration_duration.............: avg=1.44s    min=1.39s   med=1.43s    max=1.55s    p(90)=1.5s     p(95)=1.53s
     iterations.....................: 21     0.698022/s
     vus............................: 1      min=1      max=1
     vus_max........................: 1      min=1      max=1
```

http_req_duration: レスポンスタイム = latency

http_reqs: 1秒間あたりに処理できたリクエスト数 = throughput

パフォーマンス改善（インデックスの追加）

comments table への index の追加 (cf. 3-4)

```
$ docker container exec -it webapp-mysql-1 bash
# mysql -u root -proot
mysql > use isuconp;
mysql > ALTER TABLE comments ADD INDEX post_id_idx(post_id);
```

パフォーマンス改善の確認（インデックスの追加）

```
data_received..................: 678 kB 23 kB/s
data_sent......................: 1.5 kB 50 B/s
http_req_blocked...............: avg=84.8µs    min=
http_req_connecting............: avg=12.17µs   min=
http_req_duration..............: avg=1.43s     min=
  { expected_response:true }...: avg=1.43s     min=
http_req_failed................: 0.00%  ✓ 0
http_req_receiving.............: avg=475.68µs min=
http_req_sending...............: avg=32.99µs   min=
http_req_tls_handshaking.......: avg=0s        min=
http_req_waiting...............: avg=1.43s     min=
http_reqs......................: 21     0.698022/s
iteration_duration.............: avg=1.44s     min=
iterations.....................: 21     0.698022/s
vus............................: 1      min=1
vus_max........................: 1      min=1
```

```
data_received..................: 8.2 MB 276 kB/s
data_sent......................: 18 kB  595 B/s
http_req_blocked...............: avg=11.88µs   min=
http_req_connecting............: avg=1.22µs    min=
http_req_duration..............: avg=120.08ms min=
  { expected_response:true }...: avg=120.26ms min=
http_req_failed................: 0.40%  ✓ 1
http_req_receiving.............: avg=411.75µs min=
http_req_sending...............: avg=23.41µs   min=
http_req_tls_handshaking.......: avg=0s        min=
http_req_waiting...............: avg=119.65ms min=
http_reqs......................: 249    8.378812/s
iteration_duration.............: avg=120.78ms min=
iterations.....................: 249    8.378812/s
vus............................: 1      min=1
vus_max........................: 1      min=1
```

パフォーマンス改善（マルチプロセス化）

プロセス数を 1 から 4 に変更 (cf. 3-5)

```
$ emacs -nw webapp/ruby/unicorn_config.rb
$ docker compose down && docker compose build
$ docker compose up
```

スループットの改善を確かめるために、並列度を上げて負荷試験を行う。

```
# k6 run --vus 4 --duration 30s ab.js
```

パフォーマンス改善の確認（マルチプロセス化）

```
data_received..................: 6.1 MB 200 kB/s
data_sent......................: 13 kB  441 B/s
http_req_blocked...............: avg=30.24µs  min=
http_req_connecting............: avg=6.93µs   min=
http_req_duration..............: avg=636.47ms min=
  { expected_response:true }...: avg=636.47ms min=
http_req_failed................: 0.00%  ✓ 0
http_req_receiving.............: avg=515.08µs min=
http_req_sending...............: avg=44.46µs  min=
http_req_tls_handshaking.......: avg=0s       min=
http_req_waiting...............: avg=635.91ms min=
http_reqs......................: 189    6.210882/s
iteration_duration.............: avg=644.5ms  min=
iterations.....................: 189    6.210882/s
vus............................: 4      min=4
vus_max........................: 4      min=4
```

```
data_received..................: 16 MB 542 kB/s
data_sent......................: 36 kB 1.2 kB/s
http_req_blocked...............: avg=16.77µs  min=1
http_req_connecting............: avg=3.79µs   min=0
http_req_duration..............: avg=238.21ms min=2
  { expected_response:true }...: avg=238.21ms min=2
http_req_failed................: 0.00% ✓ 0
http_req_receiving.............: avg=529.68µs min=6
http_req_sending...............: avg=38.36µs  min=9
http_req_tls_handshaking.......: avg=0s       min=0
http_req_waiting...............: avg=237.64ms min=0
http_reqs......................: 500    16.780077/s
iteration_duration.............: avg=240.8ms  min=8
iterations.....................: 500    16.780077/s
vus............................: 4      min=4
vus_max........................: 4      min=4
```

k6 を使ったシナリオの記述

以下のようなシナリオをテストしたい:

1. Web アプリケーションの初期化処理
2. ユーザがログインしてコメントを投稿する処理
3. ユーザがログインして画像を投稿する処理

k6 を使ったシナリオの記述

- 初期化処理: initialize.js
- ログイン -> コメントの投稿: comment.js
- ログイン -> 画像のアップロード: postimage.js

それぞれ単独でも以下のように同様に実行できる:

```
$ k6 run (initialize|comment|postimage).js
```

統合シナリオの実行

initialize.js

```
export const options = {
  scenarios: {
    initialize: {
      executor: "shared-iterations",
      vus: 1,
      iterations: 1,
      exec: "initialize",
      maxDuration: "10s",
    },
    comment: {
      executor: "contant-vus",
      vus: 4,
      ...
    },
    postImage: {
      executor: "contant-vus",
      ...
    }
  }
}
```