СПбГУ, мат-мех, гр. 271

Владимир Милосердов

4 декабря 2016 г.

Использовался фильтр с ядром 7x7, цветные (RGB) изображения. Для тестирования сервера был реализован простой клиент:

```
1  public class SimpleClient implements Callable<Long> {
2      private InetAddress address;
3      private BufferedImage image;
4      private int port;
5
6      public SimpleClient(InetAddress address, int port, BufferedImage image) {
7          this.address = address;
8          this.image = image;
9          this.port = port;
10     }
11
12     public Long call() {
13         long startTime = System.currentTimeMillis();
14         try {
15             Socket s = new Socket();
16             s.connect(new InetSocketAddress(address, port));
17             OutputStream output = s.getOutputStream();
18
19             new ImageSender(image, output).send(0);
20
21             InputStream input = s.getInputStream();
22             byte[] magicWord = new byte[1];
23             for (;;) {
24                 if (input.read(magicWord) == -1)
25                     throw new IOException("Can't read magicWord");
26                 if (magicWord[0] == (byte) 0xFF)
27                     break;
28                 byte[] cur = new byte[4];
29                 if (input.read(cur) == -1)
```

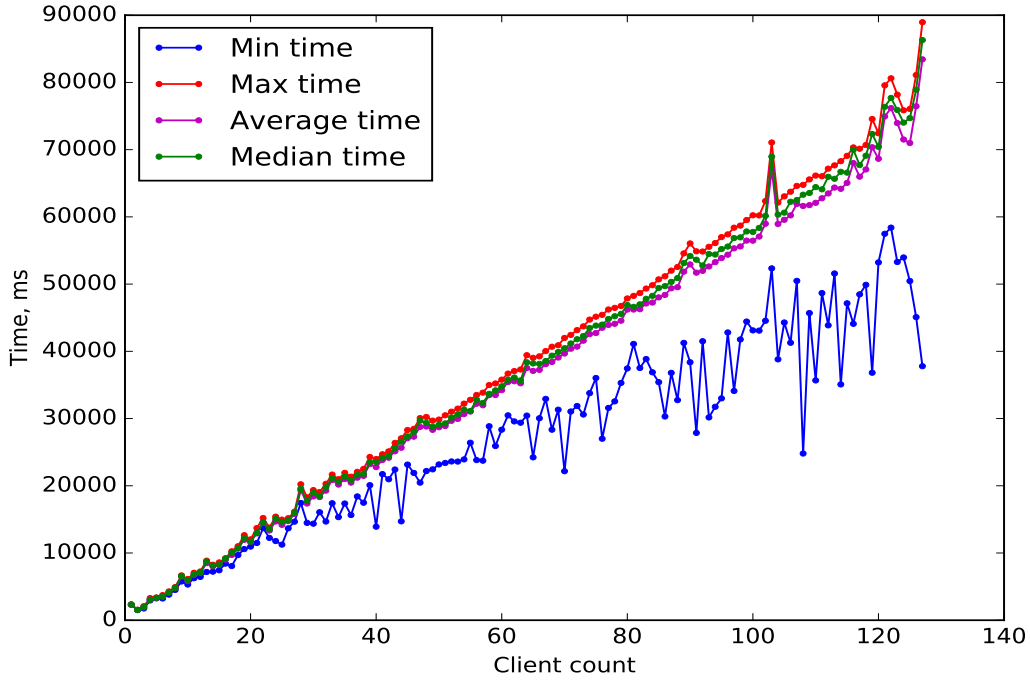
```
30                      throw new IOException("Can't read current progress");
31                  }
32
33                  // get the result and forget it
34                  new ImageReceiver(input).recv();
35              } catch (IOException e) {
36                  System.out.println("CRASHED: " + e.getMessage());
37                  return (long) -1;
38              }
39              return System.currentTimeMillis() - startTime;
40          }
41      }
```

Тестирование проходило в два этапа: на первом этапе число клиентов при фиксированном изображении **(1920x1080)** последовательно увеличивалось для нахождения количества отказа. Для этого использовался примерно следующий код:

```
1   BufferedImage image = ImageIO.read(new File("test/test.jpg"));
2   TestResult[] output = new TestResult[200];
3
4   for (int count = 1; count < 200; count++) {
5       ExecutorService executor = Executors.newCachedThreadPool();
6       ArrayList<Future<Long>> pendings = new ArrayList<>(count);
7       ArrayList<Long> results = new ArrayList<>(count);
8       for (int i = 0; i < count; i++) {
9           SimpleClient client = new SimpleClient(
10              InetAddress.getByName("127.0.0.1"), 1424, image);
11          pendings.add(executor.submit(client));
12      }
13
14      boolean failed = false;
15      for (Future<Long> cur : pendings) {
16          Long val = cur.get();
17          if (val == -1) {
18              failed = true;
19              break;
20          }
21          results.add(val);
22      }
23      executor.shutdown();
24      if (failed) {
25          System.out.println("!!! FAILED AT " + count + " !!!");
26          break;
27      }
28      output[count] = getResult(results);
```

```
29  }
30
31  try (Writer writer = new FileWriter("results.json")) {
32      Gson gson = new GsonBuilder().create();
33      gson.toJson(output, writer);
34  }
```

Далее на основе полученного на выходе файла был построен график зависимости времени выполнения от количества клиентов:



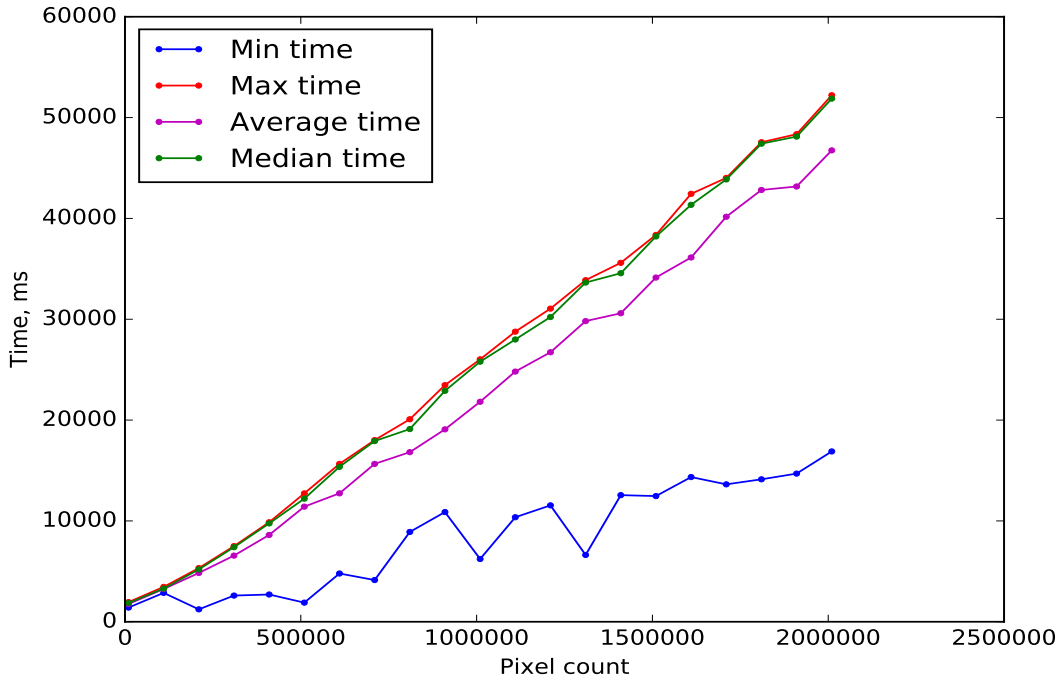
Также установлено количество одновременных клиентов, приводящих к отказу: **128**

Во второй части оценивалось время обработки запросов при фиксированном числе клиентов (это число равнялось **50**) при разных размерах изображения.
На каждом шаге простой скрипт генерировал случайное изображение нужного размера, запускал программу на Java, которая создавала 50 клиентов *SimpleClient*; затем читался результат из json-файла. Таким образом были проанализированы запуски при количестве пикселей в изображении ***от 10000 до 2100000 с шагом 100000***. Код скрипта:

```
1  #!/usr/bin/env python3
2
3  import json
4  import sys, os
```

```
5  import matplotlib.pyplot as plt
6  import time
7  from math import *
8  from PIL import Image
9  from numpy import random
10 import subprocess
11
12 # Data for plots
13 grx = [[], [], [], []]
14 gry = [[], [], [], []]
15
16 for size in range(10000, 2100000, 100000):
17     print('Testing for size =', size, '...')
18
19     width = height = ceil(sqrt(size))
20     Z = random.rand(width, height, 3) * 255
21     img = Image.fromarray(Z.astype('uint8')).convert('RGBA')
22     img.save('.tmp_img.png')
23
24     cmd = ['java', '-jar', 'test.jar', '.tmp_img.png']
25     subprocess.Popen(cmd).wait()
26
27     with open('.tmp_res.json') as data_file:
28         data = json.load(data_file)
29     os.remove('.tmp_res.json')
30     os.remove('.tmp_img.png')
31
32     for i in range(4):
33         grx[i].append(size)
34     gry[0].append(data['minTime'])
35     gry[1].append(data['maxTime'])
36     gry[2].append(data['avgTime'])
37     gry[3].append(data['medTime'])
38
39 # Plotting
40 plt.cla()
41 plt.plot(grx[0], gry[0], '-b.', label='Min time')
42 plt.plot(grx[1], gry[1], '-r.', label='Max time')
43 plt.plot(grx[2], gry[2], '-m.', label='Average time')
44 plt.plot(grx[3], gry[3], '-g.', label='Median time')
45 plt.xlabel('Pixel count')
46 plt.ylabel('Time, ms')
47 plt.legend(loc='upper left')
48 plt.savefig('plot2.pdf')
```

График:


Min time
Max time
Average time
Median time
Time, ms
Pixel count
0
10000
20000
30000
40000
50000
60000
0
500000
1000000
1500000
2000000
2500000