Verteilte Systeme

Praktikum 01 ---- Testprotokoll

# Systemvoraussetzungen

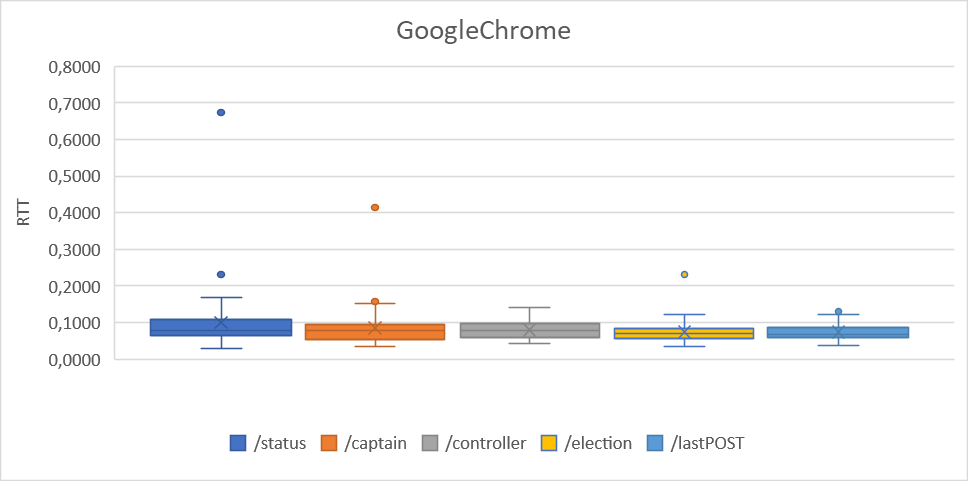
Der Controller und damit der Webserver unseres Praktikums läuft in Docker auf einem Windows Desktop, welcher mit 32 GB Arbeitsspeicher und einem Intel Core i9-9900K @3.60GHz ausgestattet ist, wobei die CPU mit 8 Kernen auf 5GHz übertacktet wurde.

Die Clients, welche auf den Webserver unseres Controllers zugreifen, werden von der gleichen Maschine aus aufgerufen, um so Netzwerkabhängigkeiten zu vermeiden. Getestet wird sowohl mit Google Chrome (v119.0.6045.160), libcurl und ApacheBench CL Utility.

Es werden Zeiten protokolliert, welche vom Webserver ins Log geschrieben werden, wenn ein Job bearbeitet wurde. Ebenso werden, wenn vorhanden, nützliche Werte vermerkt, die von den einzelnen Tools ausgegeben werden. Getestet werden immer alle einzelnen Schnittstellen am Webserver, um so schnellere oder langsamere Prozesse feststellen zu können.

# Google Chrome

Es wurden für jede Schnittstelle 50 Zugriffe nacheinander ausgeführt, die RTT, welcher der Server für die Bearbeitung der Anfrage benötigt hat, notiert und in dieser Tabelle verarbeitet.



**0,1006**

**0,0845**

**0,0800**

**0,0739**

**0,0746**

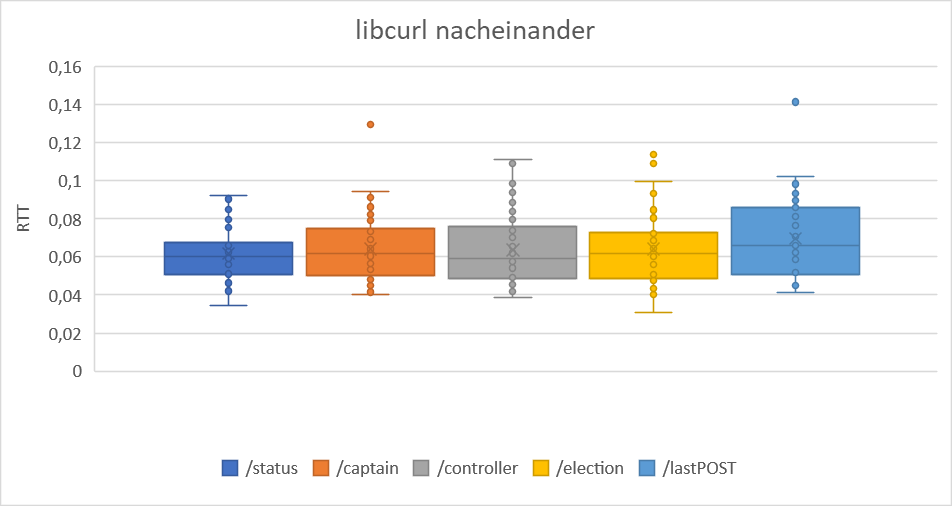
Die Werte direkt unter dem Diagramm spiegeln die Durchschnittswerte der Ergebnisse wider

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Paths** | **/status** | **/captain** | **/controller** | **/election** | **/lastPOST** |
| Size(Byte) | 67 | 55 | 79 | 76 | 25 |
| minTime(ms) | 6,13 | 5,52 | 5 | 5,15 | 6,15 |
| maxTime(ms) | 16,46 | 16,29 | 18,39 | 13,4 | 20,61 |

# Libcurl

## GET

Auch hier werden wieder 50 Zugriffe auf jede Schnittstelle nacheinander ausgeführt und protokolliert.



**0,0618**

**0,0640**

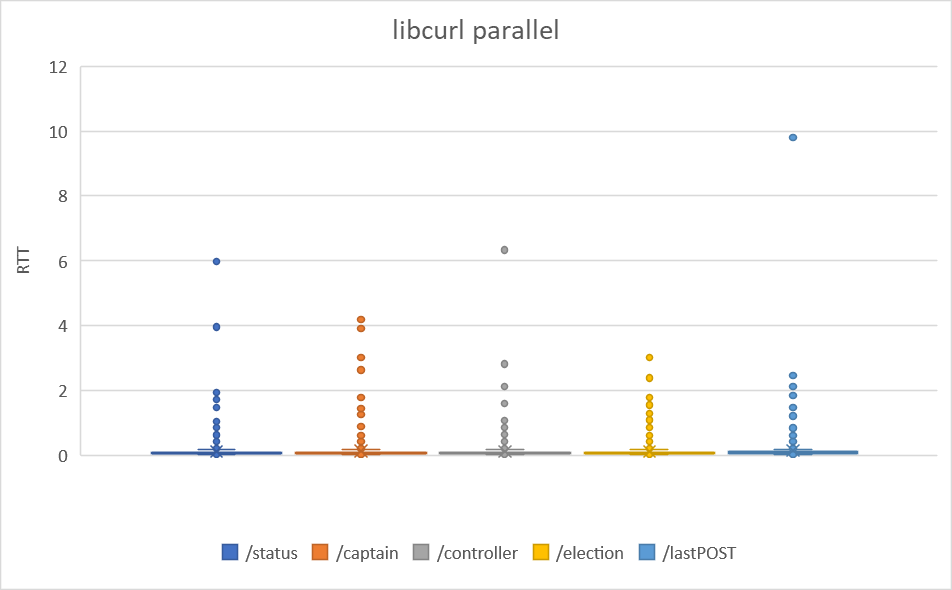
**0,0636**

**0,0640**

**0,0696**

*curl -X GET localhost:8080/PATH*

Beim nächsten Test wurde eine Datei geschrieben, die mithilfe von threads den curl-Befehl parallel ausführen kann. Getestet wurde mit 1000 threads, welche gleichzeitig auf den Server zugreifen.



**0,1175**

**0,1260**

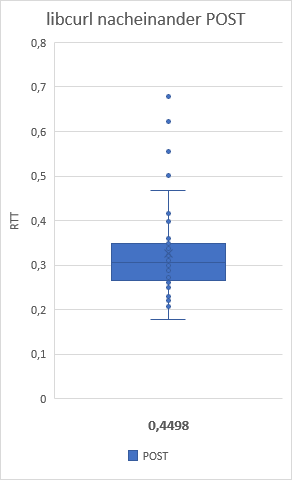
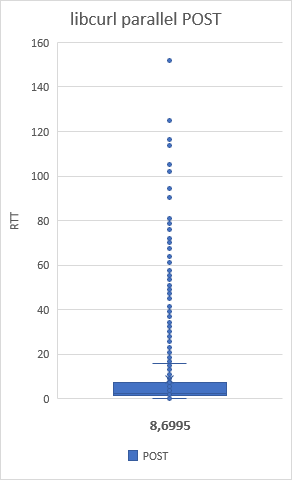
**0,1156**

**0,1207**

**0,1402**

*curl -X GET localhost:8080/PATH über /test/testThreads.cpp*

## POST

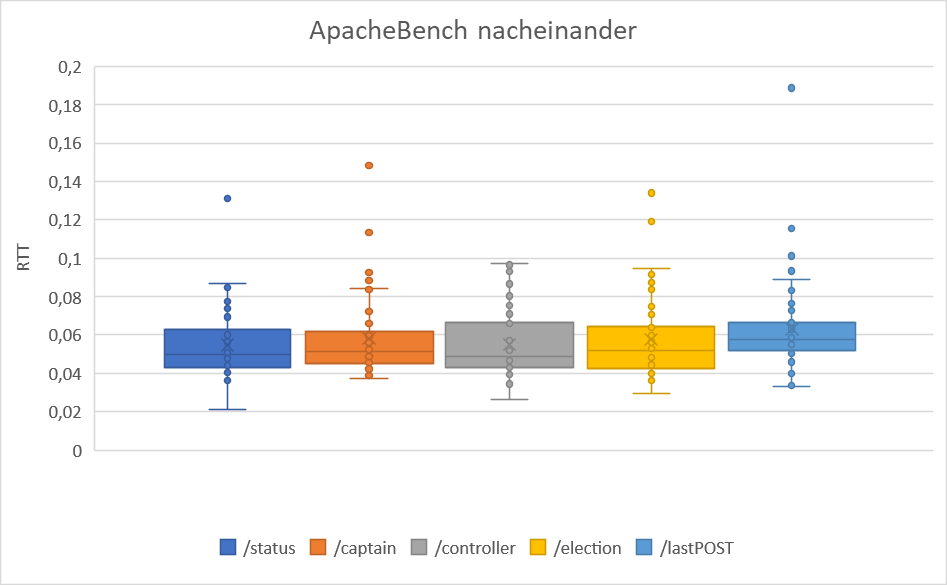


*curl -X POST -d TESTPROTOKOLL localhost:8080*

Hierbei sieht man die ersten großen Unterschiede in diesem Test. Wenn auf den Server in Reihe, also nacheinander zugegriffen wird, ist die Zeit, welche der Server zur Verarbeitung benötigt, deutlich geringer als wenn er parallele Anfragen bekommt. Dies liegt an der am Server hinterlegten Funktion, dass die Daten, welche über POST übermittelt werden, in der Datenbank, in unserem Fall in einer TXT-Datei, abgelegt werden müssen. Da dort nur von einem thread gleichzeitig in die Datei geschrieben werden kann, entsteht hier ein Bottleneck, welches in zukünftigen Iterationen verbessert werden sollte.

Beim Zugriff in Reihe ist außerdem aufgefallen, dass der erste Zugriff sehr viel länger dauert als die restlichen Zugriffe auf den Server/die Datenstruktur. Dieser hat über mehrere Tests durchschnittlich **6,5495 ms, also ca. das 15-fache,**gebraucht.

# ApacheBench



**0,1175**

**0,1260**

**0,1156**

**0,1207**

**0,1402**

*ab -n 50 -c 1 http://localhost:8080/PATH*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Paths** | **/status** | **/captain** | **/controller** | **/election** | **/lastPOST** | **POST** |
| time taken for tests(s) | 0,04 | 0,039 | 0,037 | 0,042 | 0,038 | 0,058 |
| Requests per sec(#/s) | 1239,5 | 1289,12 | 1368,63 | 1181,39 | 1310,24 | 868,76 |
| Time per request(ms) | 0,807 | 0,776 | 0,731 | 0,846 | 0,763 | 1,151 |
| Transer rate(kb/s) | 81,1 | 69,24 | 106,31 | 87,68 | 40,94 | 41,57 |
| Failed Connections | 0 | 0 | 7 | 0 | 0 | 0 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Concurrency Level** | **10** | **50** | **100** | **500** | **1000** | **1020** |
| time taken for tests(s) | 146,733 | 133,179 | 128,667 | keineDaten | 176,041 | 246,55 |
| Requests per sec(#/s) | 6815,09 | 7508,69 | 7772 | keineDaten | 5680,5 | 4055,97 |
| Time per request(ms) | 0,147 | 0,113 | 0,129 | keineDaten | 176,041 | 0,247 |
| Transer rate(kb/s) | 372,7 | 410,63 | 425,03 | keineDaten | 310,55 | 221,74 |
| Failed Connections | 0 | 0 | 0 | 34 | 337 | 342 |

Der letzte Test dieses Testprotokolls wurde mit dem Shell-Befehl der ApacheBench API durchgeführt. Hierbei wurden bei jedem Durchlauf **1.000.000** Anfragen an den Server geschickt, wobei der „Concurrency Level“ die Anzahl an parallel laufenden Anfragen widerspiegelt. Wie man sieht, kann dieser bis ca. 500 gleichzeitige Anfragen ohne Probleme übernehmen. Ab 500 jedoch werden immer wieder Verbindungen nicht akzeptiert oder nicht abgeschlossen/verschluckt. Dies spiegelt sich dann auch in allen anderen Zeiten wider.