

## Contents

|          |                           |          |
|----------|---------------------------|----------|
| <b>1</b> | <b>Einführung</b>         | <b>2</b> |
| <b>2</b> | <b>Grundlagen</b>         | <b>2</b> |
| <b>3</b> | <b>Konzept</b>            | <b>3</b> |
| <b>4</b> | <b>Verwandte Arbeiten</b> | <b>5</b> |
| <b>5</b> | <b>Zeitplan</b>           | <b>6</b> |
| <b>6</b> | <b>Referenz</b>           | <b>6</b> |

# 1 Einführung

Performanz spielt eine große Rolle in der Entwicklung von Hardware und Software. Um herauszufinden ob gewisse Computer performant genug sind, werden ihre Performanz mit sogenannten Benchmarks getestet. Mit ihnen kann man die Grenzen der Hardware testen. Aber auch nicht nur Hardware, sondern auch Virtuelle Maschinen können getestet werden, wo man die Grenzen der virtuelle CPUs oder des Hypervisor herausfinden will. [2] Sowie in Cloud Computing können solche Tests durchgeführt werden [3]. Somit gibt es ein weites Spektrum an Auswahl, wo solche Benchmarks verwendet werden. Die Grenzen könnte die Hardware (z.B CPU-Taktrate oder Bandbreitengeschwindigkeit beim Lesen oder Schreiben) oder Software (Komplexität von Algorithmen) sein. *Es werden daher standardisierten Benchmarks verwendet. Mit denen ist es möglich die Performanz verschiedener Hardware- und Softwareanbieter zu vergleichen.*

In Großkonzernen sind Performanzprobleme und nicht die funktionellen Probleme die größten Hindernisse. Service Ausfälle sind höchst kostspielig. Deshalb werden diese Benchmarks in High Performance Computing durchgeführt, um ihre Grenzen zu herauszufinden [1]. Ein Beispiel Programm für so ein Benchmark Tool ist das fio (flexible I/O Generator) von alex. In meiner Arbeit werde ich mit dem fio Tool arbeiten, um solche Performanzänderungen zu berechnen und zu analysieren. Fio ist ein Tool mit der man die Bandbreitengeschwindigkeit testen kann, z.B auf eine Datei Schreiben. Solche Tests lassen sich noch als Logdatei ausgeben. Mein Programm wird mit diesen Logs arbeiten, um mit ihnen besser arbeiten zu können oder auch besser darzustellen. Da diese zehntausende von Zeilen besitzen kann. Ziel meines Tools soll es sein, dass man den stationären Zustand analysiert.

# 2 Grundlagen

Mit dem Fio , kann man die Performanz der Bandbreitengeschwindigkeit auf einer bestimmten Hardware testen. Die I/O Geschwindigkeit wird hier in KiByte pro Sekunde angegeben. Fio selbst besitzt kein GUI sondern arbeitet nur in der Konsole. Man kann für das Programm sogenannte .fio Dateien schreiben oder auch nur in der Konsole arbeiten, um seine gewünschten Tests durchzuführen, wie z.B. random read/write auf einer Datei. Wenn man nun ein Random Read testen möchte, kann die .fio Datei so aussehen.

```
1 ; fio-rand-read.job for fiotest
2
3 [global]
4 name=rand-read # Name des Jobs
5 rw=randread # Was soll der Job testen , randread = random read
6 runtime=2s # Wie lange soll der Job laufen
7 size=2m # Groesse der Datei , m fuer Megabyte
8 write_bw_log=mytest # Name der Log-Datei
```

In der Konsole kann man sie auch als Parameter angeben (z.B für rw: "–rw=randread"). Wenn es nur ein schneller Test sein soll, würde die Konsole schon ausreichen. Zusätzlich können die Tests, oder auch Jobs genannt, Logs ausgeben, mit denen mein Tool arbeiten wird. Dieser Job oben testet das Random-Read mit einer Runtime von 2 Sekunden. Nun möchte man wissen, wann der stationäre Zustand erreicht wurde, wenn man solche Tests länger durchlaufen lässt. Der stationäre Zustand ist der Zeitpunkt, wenn es keine starken Schwankungen mehr bei der Bandbreitengeschwindigkeit im Lesen **und** **oder** Schreiben gibt. Das Schreiben und Lesen verhält sich außerdem nicht deterministisch. Das bedeutet, auch wenn man die selbe Datei mit dem selben Computer lesen/schreiben würde, wäre die Bandbreitengeschwindigkeit immer im Durchschnitt unterschiedlich. Weitere nicht deterministische Zustände könnte sogar CPU Temperatur sein, oder CPU Scaling, parallele Prozessierung [SOURCE]. Da die Geschwindigkeit sich nie konstant einem Wert nähert, soll das Programm in Zukunft die Standardabweichung errechnen, wo man selber festlegt, wie groß die Abweichung sein kann.

Die Geschwindigkeit der Bandbreite ist abhängig von der CPU und des Speichermediums, wie SATA HDD, SATA SSD oder NVMe. The main data created by our experiment is the time taken by each in-process iteration to run. Formally, this is time series data of length 2000. In this Section we explain how we use statistical changepoint analysis to enable us to understand this time series data and classify the results we see, giving the first automated method for identifying warmup. *NVMe drives are expected to be widely adopted in datacenters*. Warmup iterations are intended to bring the JVM into a steady state (e.g., execute all applicable just-in-time compilations).

### 3 Konzept

fiio besitzt ein extra Parameter, um solche Workloads als log ausgeben zu können. Workloads oder auch Jobs, sind die Performanz Tests die mit dem fio gemacht werden. Für diese Workloads möchte man noch gerne wissen was eigentlich für Werte Zustände kamen. Die Logdatei besitzt die Information. Sie besitzt die Information vom verlauf des Jobs in Millisekunden. Die jeweils 5 Werte besitzen und mit Kommas getrennt sind. Mit dem Parameter *write\_bw\_logs = [Dateiname]*, kriege ich diese Logdatei:

|   |  |
|---|--|
| 1 | \[Time, Bandwidth, data direction, Blocksize, Offset\] |
| 2 | 0, 59782, 0, 4096, 0                                   |

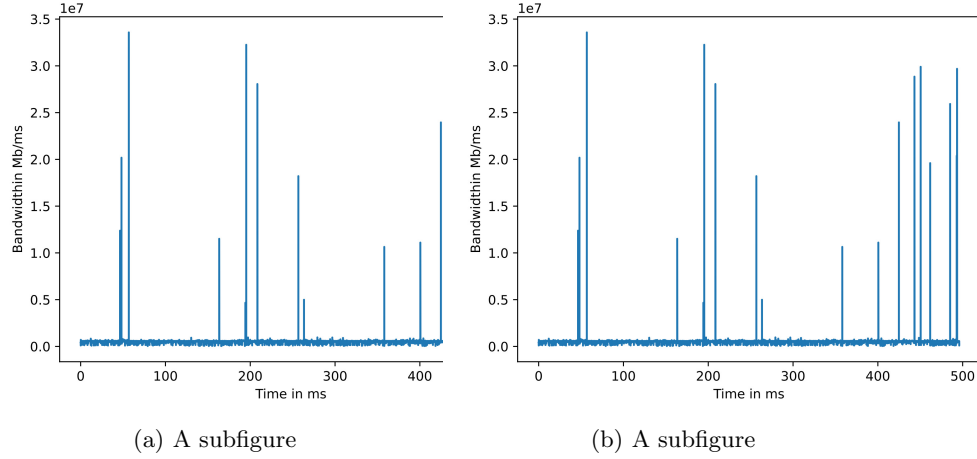


Figure 1: A figure with two subfigures

Time für die Zeit in Millisekunden (ms) die verlaufen ist, Bandwidth für die Brandweite in KiByte/s Dritter Wert für die Date direction ob gelesen (= 0) oder geschrieben (= 1) wurde ein Blocksize und ein Offset. Die Logdaten selbst sind immens lang und nicht schön lesbar. Deshalb wurde schon ein kleines Tool in Python gebaut was diese Logdaten zu einem Graph umwandelt. Die Y-Achse ist die Bandbreitengeschwindigkeit und X-Achse ist die vergangene Zeit des jeweiligen Jobs. Wenn man die Logdaten anschaut, sieht man, dass die Brandbreitengeschwindigkeit sich schneller innerhalb einer 1 ms ändert.

|    |  |          |    |                    |          |  |
|----|--|----------|----|--------------------|----------|--|
| 1  | Drei Zeilen aus der Logdatei                         |          |    |                    |          |  |
| 2  | [Time, Bandwidth, data direction, Blocksize, Offset] |          |    |                    |          |  |
| 3  | 23,  | 59782,   | 0, | 4096,              | 0        |  |
| 4  | 23,  | 43534,   | 0, | 4096,              | 0        |  |
| 5  | 23,  | 54364,   | 0, | 4096,              | 0        |  |
| 6  |  |          |    |                    |          |  |
| 7  | Erste beide Spalten                                  |          |    | -> X-Y-Koordinaten |          |  |
| 8  | X: 23  | Y: 59782 | -> | X: 23.0            | Y: 59782 |  |
| 9  | X: 23  | Y: 43534 | -> | X: 23.3            | Y: 59782 |  |
| 10 | X: 23  | Y: 54364 | -> | X: 23.6            | Y: 59782 |  |

Da man die Daten nicht verwerfen möchte, wurden Wiederholungen linear getrennt. Diese Veranschaulichung wurde auf meinem Rechner getestet mit dem Command die oben erwähnt wurde. Die logdatei wurde ausgewertet und hier als .svg Datei dargestellt:

Der Computer hat ein 11th Gen Intel(R) Core(TM) i5-11400 @ 2.60GHz und eine ADATA SWORDFISH SSD als Datenträger. Es wurde ein kleines Tool geschrieben. Die eigentliche Arbeit wird es sein, das Tool weiter auszubauen das nicht nur veranschaulicht sondern auch den stationären Zustand berechnet. Nun erkennt man aber im Graph, dass diese Spikes zu sehen sind. Diese Spikes entstehen durch nicht deterministische Faktoren. Das fio arbeitet nicht nur mit einem Thread, sondern es arbeitet Multithreaded, was zu nicht Determinismus führt. Das Programm wird in Java geschrieben. Grund dafür ist, da dort die bereits die meiste Erfahrung steckt. Eine Möglichkeit, den stationären Zustand zu ermitteln, wäre das man die Standardabweichung verwendet und errechnet, wann die Abweichung klein ist. *With improvements in storage device technology, the once negligible cost of I/O stack time has become more relevant (Foong et al. 2010; Caulfield et al. 2010). A number of studies have provided proof of the I/O software stack being the major performance bottleneck in future storage systems.* Im nächsten Graph werden die Häufigkeiten genommen für die Bandbreitengeschwindigkeit. X-Achse sind die Anzahl an Wiederholungen und Y-Achse die Geschwindigkeit.

## 4 Verwandte Arbeiten

Es wird auch auf nicht Determinismus von Computer, oder auch VMs, eingegangen [5]. Sie versucht man so gering wie möglich zu halten, da sonst bei der Auswertung nicht akkurate Werte berechnet werden können. Nun lässt sich multithreading schlecht minimieren, da es dort meist Random verläuft, aber einige Methoden gibt es. Man kann, bei VMs, erst sie erst mal warm laufen lassen [1]. Oder man versucht statistisch Methoden und arbeitet mit der Standardabweichung oder Konfident-werten Das Tool, was entwickelt werden soll, wird mit solchen Methoden arbeiten. Die Methodiken oder statistischen Herangehensweisen die genommen werden, stammen aus verschiedenen Arbeiten. Wann wird ein warmup erfolgt sein [5] oder wann der stationären Zustand erreicht wurde [5]. Dabei soll das Tool nicht für spezifische Hardware dienen und wird zur reinen Analyse von fio Daten dienen. Dies ist von dem fio [6] Tool abhängig. Another source of non-determinism comes from thread scheduling in time-shared and multiprocessor systems. Running multithreaded workloads, as is the case for most Java programs, requires thread scheduling in the operating system and/or virtual machine.

## 5 Zeitplan

## 6 Referenz

- – An Automated Approach for Recommending When to Stop Performance Tests [1] –
- Virtual Machine Performance Analysis and Prediction [2]
- A Comparative Study of High-Performance Computing on the Cloud [3]
- A Case for High Performance Computing with Virtual Machines [4]
- Virtual Machine Warmup Blows Hot and Cold [5]
- aleks/fio auf git [6]