Contents

1 Einführung

Die Performanz spielt eine große Rolle in der Entwicklung von Hardware und Software. Um herauszufinden ob gewisse Systeme eine bessere Rechenleistung besitzen als andere Systeme, werden ihre Performanz mit bestimmter Software, genannt Benchmarks, getestet. Mit ihnen kann man die Grenzen der Hardware oder auch Software testen. Ein Beispiel wo Software getestet wird, sind Virtuelle Maschinen. Hier werden die Grenzen von virtuelle CPUs oder des Hypervisor analysiert. [4] Ein weiter Bereich, wo Benchmarking genutzt wird, ist Cloud Computing [5]. Somit gibt es ein weites Spektrum an Auswahl, wo solche Benchmarks verwendet werden. Solche Grenzen könnte die Hardware, z.B CPU-Taktrate oder Brandbreitengeschwindigkeit beim Lesen oder Schreiben, aber auch Software (Komplexität von Algorithmen) sein.

In Großkonzernen sind oft Performanzprobleme die eigentlichen Hindernisse, und nicht die funktionellen Probleme. Service Ausfälle sind höchst kostspielig. Aus diesem Grund werden solche Benchmarks in High Performance Computing (HPC) durchgeführt, um ihre Grenzen zu analysieren [1]. Ein Beispiel Programm für so ein Benchmark Tool ist das axboe/fio (flexible I/O tester) [2]. In meiner Arbeit werde ich mit dem fio Tool arbeiten, um solche Performanz Änderungen zu berechnen und zu analysieren. Fio ist ein Tool mit der man die Brandbreitengeschwindigkeit vom Lesen/Schreiben testen kann. Diese Tests lassen sich als eine Logdatei ausgeben. Mein Programm wird mit diesen Logs arbeiten, um sie darstellen und analysieren zu können. Da diese Dateien zehntausende von Zeilen besitzen können. Das Ziel soll es sein, dass man den stationären Zustand berechnet. [6]

2 Grundlagen

Das Fio Tool ermöglicht das Testen auf bestimmter Hardware oder Software. 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. ein random Read auf einer Datei. Wenn man nun ein Random Read testen möchte, kann die .fio Datei wie folgt aussehen:

```
1  ; fio-rand-read.job for fio Test
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=128m # Groesse der Datei, m fuer Megabyte
8  write_bw_log=mytest # Name der Log-Datei
```

In der Konsole kann man sie auch einzeln als Parameter angeben, wenn es nur ein Test sein soll. 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 maximalen Laufzeit (runtime) von 2 Sekunden und ließt eine Datei mit einer Größe von 2 Mebibyte. Mit größeren Tests ist es möglich den stationären Zustand zu analysieren. Der stationäre Zustand ist der Zeitpunkt, wenn es keine starken Schwankungen mehr bei der Brandbreitengeschwindigkeit im Lesen oder im Schreiben gibt. Ein Hindernis bei dieser Analysierung könnten die nicht deterministischen Faktoren sein. Auch wenn man Das Lesen/Schreiben auf dem selbem System mit der selben Datei testet, ist die Brandbreitengeschwindigkeit nie die gleiche, sondern sie schwankt immer. Ursachen dafür könnten schon verschiedene CPU Temperaturen sein, aber auch CPU Scaling oder parallele Prozessierung. Da die Geschwindigkeit sich nie konstant einem Wert nähert sondern immer abweicht, soll mein Programm in Zukunft mithilfe der Standardabweichung den stationären Zustand ermitteln, Solche Abweichung könnte in wenigen Prozentbereichen liegen.

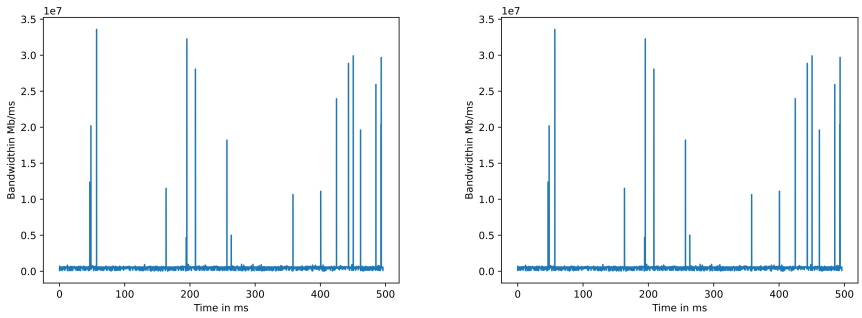
Es wird aber nicht ausreichen ein paar Tests durchzuführen und danach die Logs davon auszuwerten. Diese Logs wären nämlich nicht akkurat genug. Man muss erst einige Warmup Iteration durchführen. Erst dann werden die Werte präziser [6]. Somit hängt der stationäre Zustand mit diesen Warmups zusammen. Dies sollte aber nur ein kleines Hindernis für mein Tool sein. Ein größeres Probleme könnte sein, wenn der Warmup durchgelaufen ist, welche Logs am Ende die besten Information besitzen. [1]

3 Konzept

fio 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 Zustande 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
```

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 Brandbreitengeschwindigkeit und X-Achse ist die vergangene Zeit des



(a) A subfigure (b) A subfigure

Figure 1: A figure with two subfigures

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: 43534
10    X: 23     Y: 54364       ->      X: 23.6     Y: 54364
```

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. Wann wurde der stationäre Zustand nun erreicht oder wann ist der Warmup fertig ? F-test, which is used to test whether two variances are significantly different. Tukey HSD. Analysis of Vari-

ance (ANOVA). Mein Tool soll am Ende den stationären Zustand (z.B Warmup von VMs) berechnen können, verschiedene Logs vergleichen können mit Hilfe der Standardabweichung, Varianz oder Tukey HSD

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 Brandbreitengeschwindigkeit. 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 [3]. 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 [3]. 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 verschieden Arbeiten. Wann wird ein warmup erfolgt sein [3] oder wann der stationären Zustand erreicht wurde [3]. Dabei soll das Tool nicht für spezifische Hardware dienen und wird zur reinen Analyse von fio Daten dienen. Dies ist von dem fio 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

5.1 Einführung

- Software und Performanz Bedeutung
- Benchmarks
- VMs, Cloud Computing
- Brandbreitengeschwindigkeit
- Probleme schlechter Performanz
- fio von axboe
- Mein Tool
- Allgemeines Ziel

5.2 Grundlagen

- Erklärung fio
- Ausführung fio
- .fio Datei
- fio in der Konsole
- Logs und Jobs
- Non-determinism
- stationärer Zustand
- Standardabweichung

6 Exposee Notiz

6.1 Konzept

- Tool mit den Eigenschaften:
- Non-determinism
- stationärer Zustand
- Standardabweichung
- Varianz
- (Tukey HSD)

- Warmup VMs
- Sums of Squares (SSA)

References

[1] Hammam M. Alghmadi, Mark D. Syer, Weiyi Shang, and Ahmed E. Hassan. An automated approach for recommending when to stop performance tests. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 279–289, 2016.

[2] Jens Axboe. Github—axboe/fio: Flexible i/o tester. *Retrieved Nov*, 10:2022, 2021.

[3] Edd Barrett, Carl Friedrich Bolz-Tereick, Rebecca Killick, Sarah Mount, and Laurence Tratt. Virtual machine warmup blows hot and cold. *Proc. ACM Program. Lang.*, 1(OOPSLA), October 2017.

[4] Yuegang Li, Dongyang Ou, Congfeng Jiang, Jing Shen, Shuangshuang Guo, Yin Liu, and Linlin Tang. Virtual machine performance analysis and prediction. In *2020 International Conference on Communications, Computing, Cybersecurity, and Informatics (CCCI)*, pages 1–5, 2020.

[5] Aniruddha Marathe, Rachel Harris, David K. Lowenthal, Bronis R. de Supinski, Barry Rountree, Martin Schulz, and Xin Yuan. A comparative study of high-performance computing on the cloud. In *Proceedings of the 22nd International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '13, page 239–250, New York, NY, USA, 2013. Association for Computing Machinery.

[6] David Georg Reichelt, Stefan Kühne, and Wilhelm Hasselbring. Automated identification of performance changes at code level. In *2022 IEEE 22nd International Conference on Software Quality, Reliability and Security (QRS)*, pages 916–925, 2022.