Contents

1 Einführung

Performanz spielt eine große Rolle in der Entwicklung von Hardware. Um herauszufinden ob die Hardware performant genug ist, werden Workloads verwendet um die Grenzen der Hardware zu testen. Aber auch nicht nur Hardware, sondern Virtuelle Maschinen können getestet werden, wo virtuelle CPUs oder der Hypervisor getestet werden. [2] Sowie in Cloud Computing können solche Tests durchgeführt werden [3]. Somit gibt es ein weites Spektrum an Auswahl, wo solche Tests wichtig sind. *Die Performanz eines Systems kann dabei sowohl durch Probleme in der Soft- ware, CPU-Zeit oder andere Hardwareressourcen begrenzt werden. Durch die standar- disierten Benchmarks wird es möglich, die Performanz verschiedener Hardware- und Softwareanbieter zu vergleichen.*

Die größten Probleme in den Großkonzernen sind Performanzprobleme und nicht die funktionellen Probleme. Service Ausfälle kosten sind höchst kostspielig. Deshalb werden in High Performance Computing viele Tests durchgeführt, um ihre Grenzen zu analysieren [1]. In meiner Arbeit werde ich mit dem fio Tool arbeiten, um solche Performanzänderungen zu berechnen und zu analysieren. Fio (flexible I/O Generator) ist ein Tool mit der man die Hardware-Performanz testen kann, z.B die Brandweite beim Schreiben einer Datei, und diesen Performanz-Test als log Daten ausgeben kann. Mein Programm wird mit diesen Logs arbeiten, um mit ihnen besser arbeiten zu können oder auch besser darzustellen. Ziel meines Tools soll es sein, dass man den stationären Zustand analysiert.

2 Grundlagen

Mit dem Fio , kann man die Performanz auf einer bestimmten Hardware testen. I/O Geschwindigkeit wird in hier KiByte/Sekunde angegeben.

fio besitzt kein GUI sondern arbeitet nur in der Konsole. Man kann für das Programm fio-Dateien schreiben oder auch in der Konsole arbeiten, um seine gewünschten Task zu testen, wie random read/write auf einer Datei. Wenn man 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 gebt man sie als Parameter an, wenn es nur ein schneller Test sein soll. Zusätzlich können die Tests, oder auch Jobs genannt, Logs ausgeben. Mit diesen Logs wird mein Tool arbeiten. 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 Brandweite im Lesen und 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 Brandbreitengeschwindigkeit immer im Durchschnitt unterschiedlich. Weitere nicht deterministische Zustände könnte sogar CPU Temperatur sein, oder CPU Scaling, parallele Prozessierung [SOURCE]. In Zukunft soll das Programm diese mit der Standardabweichung selber ausrechnen.

Die Geschwindigkeit der Brandbreite 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., exe- cute all applicable just-in-time compilations).

3 Konzept

Mein Tool soll mit den Logs vom fio Tool arbeiten.
fio besitzt ein extra Parameter, um solche Workloads als log ausgeben zukönnen. Workloads oder auch Jobs, sind die Performanz Tests die mit dem fio gemacht werden. Für diese Workload möchte man noch gerne 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 jeweiligen Jobs. Wenn man die Logdaten anschaut, sieht man, dass die Brandbreitengeschwindigkeit sich schneller innerhalb einer 1 ms ändert.

```
\[Time, Bandwidth, data direction, Blocksize, Offset\]
23,      59782,    0,              4096,     0
23,      43534,    0,              4096,     0
23,      54364,    0,              4096,     0

X: 23      Y: 59782    ->      X: 23.0      Y: 59782
X: 23      Y: 43534    ->      X: 23.3      Y: 59782
X: 23      Y: 54364    ->      X: 23.6      Y: 59782
```

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

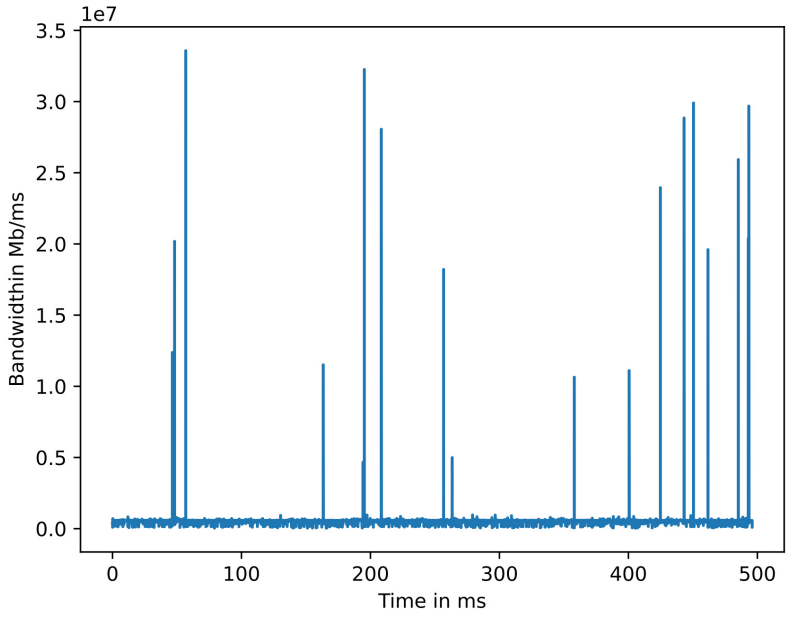

Figure 1: Plot tool Ausgabe mit Log Datei

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. 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.

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 beim Auswertung nicht akkurate Werte berechnet werden können. Um solche Fehler zu minimieren versucht man erst die VMs erst mal warmzulaufen [1]. Man wird mit Werten rechnen wie F1-Score oder relativen Standardabweichung. Das Tool soll noch mehr Funktion besitzen und umfangreicher mit den Logdaten arbeiten. Die Methodiken die genommen werden, stammen aus verschieden Arbeiten. Wann wird ein warmup erfolgt sein [5] oder wann der stationären Zustand erreicht wurde [5]. Dabei soll das Tool, was in java gebaut wird, nicht für spezifische Hardware dienen und wird zur Analyse von fio dienen. Dies ist von dem fio [6] Tool abhängig

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]