

Institut für Informatik
Fakultät für Mathematik und Informatik
Abteilung Datenbanken

Entwicklung eines Analysetools, das den stationären Zustand aus den Logs des Fio-Benchmark-Tools ermittelt

Exposé

vorgelegt von:
xxxx xxxxx

Matrikelnummer:
xxxxxxx

Betreuer:
Prof. Dr. Erhard Rahm
XXXXX

© xxxx

Dieses Werk einschließlich seiner Teile ist **urheberrechtlich geschützt**. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtsgesetzes ist ohne Zustimmung des Autors unzulässig und strafbar. Das gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen sowie die Einspeicherung und Verarbeitung in elektronischen Systemen.

Contents

| | | |
|----------|---------------------------|----------|
| 1 | Einführung | 3 |
| 2 | Grundlagen | 3 |
| 3 | Konzept | 5 |
| 4 | Verwandte Arbeiten | 7 |
| 5 | Zeitplan | 8 |

Abstract

Mit Benchmarks kann man die Performance von Computersystemen, wie etwa die Rechenleistung und die I/O-Geschwindigkeit, messen. Diese Messungen können stark oder schwach schwanken, je nachdem, ob der Rechner gerade erst mit den Messungen begonnen hat oder schon länger läuft. In dieser Arbeit wird mit dem Benchmark-Tool Fio gearbeitet, und es wird ein Analysetool entwickelt, das den Zustand ermittelt, in dem die Schwankungen der Messungen am geringsten sind. Dabei wird statistische Analyse angewendet, um diesen Zustand zu bestimmen. Das Programm wird in Java geschrieben.

1 Einführung

Die Performanz eines Rechners ist die Leistungsfähigkeit und Effizienz bei der Ausführung von Aufgaben. Um herauszufinden, welche ausgewählten Systeme die beste Performanz liefert, werden Benchmarks verwendet. Mit ihnen kann man die Rechenleistung oder auch Lese-/Schreibgeschwindigkeit der Hardware oder auch Software mit standardisierten Tests messen [6]. Ein Beispiel, bei dem Software getestet wird, sind virtuelle Maschinen. Hier werden die Grenzen von virtuellen CPUs oder des Hypervisor analysiert [9]. Auch im Cloudcomputing Bereich wird Benchmarking betrieben, um auch Performanz zwischen verschiedenen Cloudplattformen vergleichen zu können. Eine Metrik wäre hier die Turnaround-Zeit, also die Zeit, die ein Prozess benötigt, um vollständig durchzulaufen. [10]. Somit gibt es ein weites Spektrum an Auswahl von Systemen, wo solche Benchmarks verwendet werden können, um Grenzen zu testen und zu vergleichen. Solche Grenzen könnten die Hardware (z. B. CPU-Taktrate oder Bandbreitengeschwindigkeit beim Lesen oder Schreiben), aber auch Software (Virtuelle Maschinen) sein. Und mit der Verbesserung von solchen Speichermedien, wie HDD zu SSD, und CPU-Leistung wird die I/O Geschwindigkeit immer relevanter. Studien haben schon bewiesen, dass dies eine Ursache für Bottlenecks in der Performanz ist [13].

In Großkonzernen sind oft Performanzprobleme die eigentlichen Hindernisse, und nicht die funktionellen Probleme. Serviceausfälle sind höchst kostspielig und sollten minimiert werden. [3]. Aus diesem Grund wird Benchmarking in Hochleistungsrechner (High Performance Computing - HPC) betrieben, um ihre Performanz zu ermitteln. Ein Beispielprogramm für so ein Benchmark Tool ist das Fio (flexible I/O tester) [4]. Fio ist ein Tool, mit dem man die Bandbreitengeschwindigkeit 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 oft aus zehntausenden von Zeilen bestehen. In meiner Arbeit werde ich mit dem fio Tool arbeiten, um solche Performanceänderungen zu analysieren und den stationären Zustand zu ermitteln. Ein Zustand, wo die Rechenleistung oder Lese-/Schreibgeschwindigkeit sich nicht mehr beim Messen stark ändert [5].

2 Grundlagen

Das Fio Tool ermöglicht das Testen auf bestimmter Hardware oder Software. Die I/O-Geschwindigkeit wird hier in Mebibyte pro Sekunde angegeben. Das Fio bietet auch die Möglichkeit diese Einheit zu ändern. Das Programm selbst besitzt kein GUI, sondern arbeitet nur in der Konsole. Man kann für das Programm sogenannte .fio Dateien schreiben. Oder man arbeitet direkt in der Konsole, um seine gewünschten Tests durchzuführen. Wenn man nun ein Random Read testen möchte, kann die .fio Datei wie folgt aussehen (Listing 2):

```

1 ; fio-rand-read.job for fio Test
2
3 [global]
4
5 name=rand-read # Name des Jobs
6 rw=randread # Was soll der Job testen, randread = random read
7 runtime=2s # Wie lange soll der Job laufen
8 size=128m # Groesse der Datei, m fuer Megabyte
9 write_bw_log=mytest # Name der Log-Datei

```

Listing 1: Beispiel für eine .fio Datei

In der Konsole kann man sie auch einzeln als Parameter angeben, wenn es nur ein Test sein soll. Diese Tests, oder auch Jobs genannt, geben Logs aus, mit denen mein Tool arbeiten wird. Dieser Job oben testet das Random Read mit einer Laufzeit (Runtime) von 2 Sekunden und liest eine Datei mit einer Größe von 128 Megabyte. 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 Bandbreitengeschwindigkeit im Lesen oder im Schreiben gibt [5]. Der Nichtdeterminismus wird dabei ein Hindernis sein. Auch wenn man das Lesen/Schreiben auf demselben System mit derselben Datei testet, ist die Bandbreitengeschwindigkeit im Durchschnitt nie die gleiche. Sie wird immer abweichen. Ursachen dafür könnten schon verschiedene CPU-Temperaturen sein, aber auch CPU-Scaling oder parallele Prozessierung. [7]. Das fio selbst arbeitet nicht nur mit einem Thread. Sondern es arbeitet mit Multithreads, was nicht deterministische Zustände hervorrufen kann. Da sich die Geschwindigkeit nie konstant einem Wert nähert, sondern stets abweicht, soll mein Programm in Zukunft mithilfe der Standardabweichung den stationären Zustand ermitteln und dabei den Nichtdeterminismus berücksichtigen. 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 Warm-up-Iterationen durchführen. Erst dann werden die Werte am Ende präziser [5]. Somit hängt der stationäre Zustand, der errechnet werden soll, mit diesen Warmups zusammen. Dieses Problem wird aber nur ein Hindernis sein. Ein größeres Problem wird es sein - wenn der Warmup durchgelaufen ist - welche Logs die besten Informationen besitzen. Die Auswahl der Logs und deren Auswertung könnten signifikante Unterschiede beinhalten, die die Evaluierung verändern könnten [3].

3 Konzept

Das fio Tool bekommt den Parameter `-write_bw_logs = [Dateiname]` dazu, um die Logs als Datei auszugeben. Diese Logs geben den kompletten Verlauf des Jobs wieder. Die ersten Zeilen des Logs könnten so aussehen, wie in Listing 2:

```

1 [Time, Bandwidth, data direction, Blocksize, Offset]
2 0, 59782, 0, 4096, 0
3 0, 54353, 0, 4096, 0
4 1, 45545, 0, 4096, 0

```

Listing 2: Erste Zeilen des Logs (Bezeichnungen sind nicht im Log enthalten)

Time für die Zeit in Millisekunden (ms) die verlaufen ist, Bandwidth für die Bandbreitengeschwindigkeit in Kibibyte/s, der dritte Wert für die data direction, ob gelesen (= 0) oder geschrieben (= 1) wurde und ein Blocksize und ein Offset. Die Logdaten selbst sind immens lang und nicht schön lesbar. Deshalb wurde schon ein kleines Tool in Python programmiert, was diese Logdaten zu einem Graph umwandelt. Das Tool nimmt die Zeit als X-Achse und die Bandbreitengeschwindigkeit als Y-Achse. Wenn man sich die Logdaten als Text nochmal anschaut, sieht man, dass die Bandbreitengeschwindigkeit sich schneller innerhalb einer 1 ms ändert. Das Python-Tool umgeht das Problem so, dass die Werte, die mehrfach doppelt vorkommen, linear auftrennt. Unten wird es nochmal veranschaulicht (Listing 3), wenn die Zeit sich dreimal wiederholt, mit 23ms.

```

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

```

Listing 3: Aufsplitten der Zeit um ein Grafik zu bilden

In Figure 1 ist einmal die Bandbreitengeschwindigkeit pro Millisekunde dargestellt (a) und in der anderen Abbildung (b) zeigt es die Häufigkeiten, wie oft diese Bandbreitengeschwindigkeit erreicht wurde. Das fio wurde wie oben in Listing 1 ausgeführt.

Der Computer, der verwendet wurde, hat ein 11th Gen Intel(R) Core(TM) i5-11400 @ 2.60GHz Prozessor und eine ADATA SWORDFISH 1TB SSD als Datenträger. Das kleine Tool diente mehr zum Einstieg. Die eigentliche Arbeit wird es sein, das Tool weiter auszubauen, das nicht nur veranschaulicht,

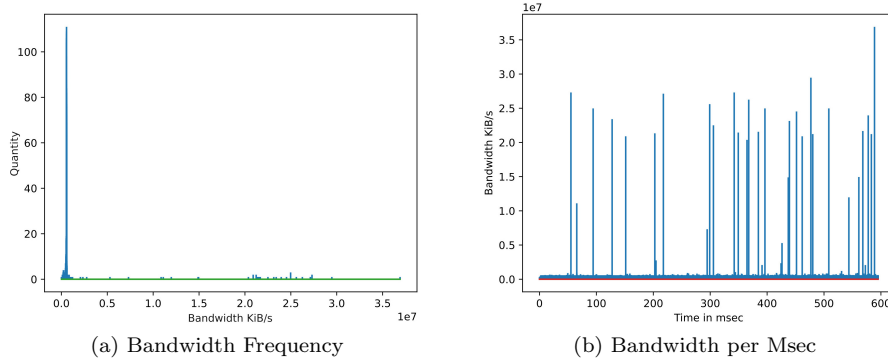


Figure 1: fio Job: randread Bandbreite und Frequenz/Quantität

sondern auch den stationären Zustand berechnet. Das Programm wird in Java geschrieben. Grund dafür ist, da dort bereits die meiste Erfahrung steckt. Eine Möglichkeit, den stationären Zustand zu ermitteln, wäre, dass man die Standardabweichung verwendet und errechnet, wann die Abweichung klein genug ist und zusätzlich Konfidenzintervalle benutzt, um nicht determinismus mitzubeachten.

Eine weitere Berechnung, die durchgeführt wird, ist die Verwendung der Varianz oder auch Varianzanalyse - *Analysis of Variance* (ANOVA) [7]. Da der stationäre Zustand nicht eindeutig bestimmt werden kann, sollen diese statistischen Methoden dabei helfen. Mit ANOVA wird untersucht, ob die berechneten Mittelwerten aus den verschiedenen Logdaten sich statistisch signifikant unterscheiden. [11] Mit statistisch signifikant ist hier gemeint, ob zwischen den durchgelaufenen Tests ein Zusammenhang besteht. Um aber zwischen einzelnen Tests unterscheiden zu können, lässt sich ANOVA mit Tukey HSD (Honestly Significant Difference) und dem multiplen t-Test erweitern, um signifikante Unterschiede bei paarweisen Messungen zu analysieren.

Der Tukey-HSD-Test verwendet die Differenz zwischen zwei Mittelwerten, um ihre statistische Signifikanz zu ermitteln. Der HSD-Wert wird mittels mittlere quadratische Abweichung (MSE) berechnet und mit der Anzahl der Gruppen die getestet wurden. Die minimalste Differenz wird als HSD bezeichnet. Wenn aber die Differenz der Mittelwerten aus den Tests größer als der HSD-Wert ist, sind diese Werte statistisch signifikant. Zusätzlich versucht der Tukey HSD Test die Fehlerrate von Falsch-Positiven-Annahmen, oder auch Typ-I-Fehler, zu korrigieren. Diese Annahmen, die aussagen, dass die statistische Signifikanz vorhanden ist, obwohl das nicht der Fall ist [1]. Die t-Tests arbeiten ähnlich wie die Tukey HSD Tests. Der Unterschied besteht darin, dass sie die Fehler-rate nicht anpassen [12] [2]. So soll mein Tool mit den Logdateien statistisch arbeiten.

4 Verwandte Arbeiten

In der Arbeit von Barrett et al [5]. wurde tiefer auf Nichtdeterminismus von VMs eingegangen. Dort wurde versucht, nicht deterministischen Faktoren so gering wie möglich zu halten, da sonst der stationäre Zustand nicht erfolgen kann, auch wenn gleiche Workloads wiederholt durchlaufen werden. Und etwas andere Hardware und Betriebssysteme haben wenig Einfluss auf die Zeit von Warmups. Li et al. [9] arbeiteten ebenfalls mit VMs und testeten die Performanz von diesen Maschinen mit verschiedener Anzahl an virtuellen CPUs. Und eine große Anzahl an VCPUs erhöht nicht zwingend die Performanz. Georges et al [8]. verwendet statistische Methoden, wie Varianzanalyse und Tukey HSD, um eine akkurate Auswertung von den Experimenten berechnen zu können. Sie arbeiten ebenfalls mit VMs und VM Warmups. Gründe für nicht deterministisches Verhalten in VMs könnte Garbage Collection, Thread Scheduling oder auch Just-in-Time Compilation/Optimization sein [7]. AlGhmadi et al. erarbeiteten, wann Performanztests nach langem Testen redundant werden und wie viele Tests genug sind.

5 Zeitplan

--

References

- [1] Tukeys hsd post hoc analyse nach anova. <https://fastercapital.com/de/inhalt/Tukey-s-HSD--Post-Hoc-Analyse-nach-ANOVA.html>. Zugriff am 11. November 2024.
- [2] Tukey’s range test. https://en.wikipedia.org/wiki/Tukey%27s_range_test. Zugriff am 11. November 2024.
- [3] 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.
- [4] Jens Axboe. Github—axboe/fio: Flexible i/o tester. Retrieved Nov, 10:2022, 2021.
- [5] 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.
- [6] Isabelle Bruno. *Benchmarking*, pages 363–368. Springer Netherlands, Dordrecht, 2014.
- [7] Andy Georges, Dries Buytaert, and Lieven Eeckhout. Statistically rigorous java performance evaluation. *SIGPLAN Not.*, 42(10):57–76, October 2007.
- [8] Andy Georges, Dries Buytaert, and Lieven Eeckhout. Statistically rigorous java performance evaluation. In *Proceedings of the 22nd Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications*, OOPSLA ’07, page 57–76, New York, NY, USA, 2007. Association for Computing Machinery.
- [9] 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.
- [10] 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.

- [11] Novustat. Post hoc test: Signifikante unterschiede finden und erklären. <https://novustat.com/statistik-blog/post-hoc-test-signifikante-unterschiede-finden-und-erklaren.html>. Zugriff am 11. November 2024.
- [12] Paul Schmidt. Mittelwertvergleich nach anova. https://schmidtpaul.github.io/crashcouRse/appendix_posthoc.html. Zugriff am 12. November 2024.
- [13] Qiumin Xu, Huzefa Siyamwala, Mrinmoy Ghosh, Tameesh Suri, Manu Awasthi, Zvika Guz, Anahita Shayesteh, and Vijay Balakrishnan. Performance analysis of nvme ssds and their implication on real world databases. In *Proceedings of the 8th ACM International Systems and Storage Conference, SYSTOR '15*, New York, NY, USA, 2015. Association for Computing Machinery.