Vergleich verschiedener Filesysteme

Ausarbeitung zur Veranstaltung der Betriebssysteme

Aaron Winziers - 1176638

Lehrstuhl für Systemsoftware und Verteilte Systeme
Universität Trier
15.03.2020

1 Einleitung

Im Rahmen der Vorlesung “Betriebssysteme” im Wintersemester 2019/2020 sollte als abschließende Arbeit ein Vergleich mehrerer Filesystemen durchgeführt werden. Hierbei war die Aufgabenstellung weit ausgelegt, was die Möglichkeit zur eigenen Interpretierung erlaubte.

Diese Ausarbeitung behandelt ein Vergleich der Transfergeschwindigkeiten zwischen verschiedener Filesystemen auf USB-Speichersticks sowie auf Solid-State-Drives (SSDs). Insbesondere wird sowohl auf die Testweise eingegangen und wie diese konzipiert und durchgeführt wurde als auch auf ihre Ergebnisse.

2 Konzept des Versuchs

Die Idee hinter dieser Ausarbeitung war nicht die Filesysteme unabhängig von einander zu überprüfen, da dies schon vielfältig geschehen ist und diese Daten in großen Maßen im Netz zu finden sind. Stattdessen wurde die Performance in der ”Zusammenarbeit” zwischen den Filesystemen getestet. Das heißt, wie performant kann von einem Filesystem auf das andere Kopiert werden.

In diesem Rahmen wurden Filesysteme File Allocation Table (FAT) und Flash-Friendly File System (F2FS) getestet. Dabei sollten die Partitionen auf USB-Sticks erstellt werden. Der Hintergrund hierbei war es, die Performance in einem normalem Gebrauchsfall eines Nutzers zu testen.

Fourth Extended Filesystem (ext4)

Ursprünglich war geplant auf zwei identische USB-Sticks die Partitionen zu erstellen, jedoch trat ein Fehler auf der zu einem Hardwarefehler führte der die Folge hatte, dass eins der beiden Sticks nicht mehr nutzbar war. Aus diesem Grund mussten leider beide Partitionen auf einem einzelnen USB-Stick erstellt werden. Hierbei handelte es sich um einen Stick mit einer USB 3.0 Schnittstelle, 64GB Speicherplatz, und nach Angaben des Herstellers einer Lesegeschwindigkeit von bis zu 100 MB/s und einer Schreibgeschwindigkeit von bis zu 40 MB/s.

Weiterhin sollte nicht nur die Kopiergeschwindigkeit zwischen den USB-Partitionen getestet werden, sondern auch zwischen dem USB-Stick und auf der im Rechner eingebaute Festplatte. Somit konnten die gängigsten Anwendungsfälle von USB-Sticks abgedeckt werden. Auf der Festplatte befand sich eine Fourth Extended Filesystem (ext4) Partition. Ext4 wurde verwendet weil esdas standard-Filesystem von Ubuntu ist und Ubuntu

zur Zeit auf dem Rechner installiert war.

Die F2FS und FAT Partitionen auf dem Stick hatten jeweils eine Größe von 9GB und Festplatte im Rechner war eine Solid-State-Drive (SSD) mit 500GB Speicherplatz.

Um die Performance tatsächlich zu testen wurden eine Menge von Dateien mit verschiedenen Größen angelegt und diese wurden dann mehrmals hin und her kopiert zwischen den Partitionen und die Dauer der Kopiervorgänge wurde aufgenommen. Die Größen der Dateien und die Reihenfolge in dem sie kopiert wurden wurde festgelegt.

Die kleinste Datei war 4MB groß und die Dateien verdoppelten sich immer in ihrer Größe bis zu 2GB. Ursprünglich wurde geplant noch kleinere Dateigrößen zu verwenden, jedoch wurde in der Entwicklung des Testkonzepts erkannt, dass die Kopiervorgänge so schnell geworden sind dass das Testprogramm keine nutzbare Daten mehr produzieren konnte. Weiterhin wurde im Interesse der Zeit entschieden keine Dateien größer als 2GB zu verwenden, da die Vorgänge sonst zu lange gedauert hätten. Schon bei 2GB dauerte das Kopieren mancher vereinzelte Dateien schon mehrere Minuten.

Die Dateien wurden jeweils in aufsteigender als auch absteigender Reihenfolge. Ziel dieses Vorgehens ist es, zu versuchen Stärken oder Schwächen der Kopierrichtungen mit einer gewissen Systematik aufzudecken.

Überlegt wurde auch noch eine Menge von Dateien zufälliger Größe zu verwenden, oder in einer zufälligen Reihenfolge die Dateien zu verschieben. Es wurde dagegen entschieden, da es das Erkennen von Mustern in der Performance gegebenenfalls erschweren könnte.

3 Test Programme

Sowohl die Generation der verwendeten Dateien als auch das Sammeln der Daten erfolgt mittels Bash-Skripte.

3.1 Datei Generation

Wie in Listing 1 gesehen werden kann, wurden mittels `dd` (Data Duplicator) die Dateien mit zufälligen Daten gefüllt. Dass `dd` verwendet wurde statt andere Möglichkeiten wie `fallocate` oder `truncate` Dateien anzulegen hat hier keine weitere Begründung.

Die Dateien wurden durch ihre Größen gekennzeichnet. Obwohl Zeilen 7 bis 9 des Skripts letztendlich nicht gebraucht wurden, sie wurden im Skript gelassen um der

Vollständigkeit zu dienen und um eine Umentscheidung bezüglich der kleineren Dateigrößen zu ermöglichen und zu vereinfachen.

```
1  #! /bin/bash
2  for n in {12..22}; do
3      if [ $n -ge 20 ]; then
4          dd if=/dev/urandom of=$((2 ** ($n % 20) ))GB.bin bs=1024
               ↪ count=$(( 2 ** $n ))
5      elif [ $n -ge 10 ]; then
6          dd if=/dev/urandom of=$((2 ** ($n % 10) ))MB.bin bs=1024
               ↪ count=$(( 2 ** $n ))
7      else
8          dd if=/dev/urandom of=$((2 ** ($n) ))KB.bin bs=1024
               ↪ count=$(( 2 ** $n ))
9      fi
10 done
```

Listing 1: Skript um Dateien zu erzeugen

3.2 Daten Sammeln

Da es in diesem Fall um eine Messung ging für die es kein bereits existierendes Tool gab musste ein Programm geschrieben werden um die Daten zu erfassen.

Zunächst nahm das Skript zwei Parameter an. Erstens der Ordner in dem die zu kopierende Dateien lagen, und zweitens der Ordener in dem die Dateien kopiert werden sollen.

Der Kopiervorgang wurde mittels cp durchgeführt, da es im Gegensatz zu anderen gängigen Kopiertools wie rsync keine weiteren algorithmische verfahren anwendet. Da das Ziel war die Filesysteme zu testen und nicht die Algorithmen hinter dem Kopieren war dies naheliegend. Es wurde auch dagegen entschieden die Dateien mit mv zu verschieben um die Verarbeitung der Daten zu vereinfachen in dem die Daten von einer Kopierrichtung nicht mit den einer andern vermischt wurden.

Um die Zeit der individuellen Kopiervorgänge zu messen wurde time verwendet. Diese Daten wurden dann in eine temporäre Datei gespeichert die dann der vollständigen angehängt wurde. Die Umsetzung dieses Vorgangs wurde zugegebenermaßen unschön gelöst und eine Lösung mittels z.B. **awk** wäre möglich gewesen, jedoch fehlte mit den Alternativen ausreichende Erfahrung um sie einsetzen zu können.

Nachdem alle Dateien kopiert wurden und die Daten gespeichert wurden die Dateien wieder aus dem Zielordner gelöscht und der gesamte Vorgang ist wiederholt durchgeführt wurden. Dies passierte zehn Mal pro Kombination von Filesystemen.

```
#! /bin/bash
for i in {0..9}; do
    date +%T > temp.csv
    echo $1 >> temp.csv
    echo $2 >> temp.csv
    for n in {12..21}; do
        if [ $n -ge 20 ]; then
            time -f "%e" cp $1$(( 2 ** ($n % 20) ))GB.bin $2
                ↪ 2>> temp.csv
        elif [ $n -ge 10 ]; then
            time -f "%e" cp $1$(( 2 ** ($n % 10) ))MB.bin $2
                ↪ 2>> temp.csv
        else
            time -f "%e" cp $1$(( 2 ** ($n) ))KB.bin $2
                ↪ 2>> temp.csv
        fi
    done
    rm $2*
    mv data.csv tempdata.csv
    paste -d, tempdata.csv temp.csv > data.csv
    rm temp.csv tempdata.csv
done
```

Listing 2: Verschiebung mit wachsender Größe

Die Struktur des Skripts zum Erfassen der Daten mit einer Reihenfolge mit absteigenden Dateigrößen ist analog zu der aus Listing 2. Das Skript ist in Listing 3 abgebildet.

```
#! /bin/bash
for i in {0..9}; do
    date +%T > temp.csv
    echo $1 >> temp.csv
    echo $2 >> temp.csv
    for n in {21..12..-1}; do
        if [ $n -ge 20 ]; then
            time -f "%e" cp $1$(( 2 ** ($n % 20) ))GB.bin $2
                ↪ 2>> temp.csv
        elif [ $n -ge 10 ]; then
            time -f "%e" cp $1$(( 2 ** ($n % 10) ))MB.bin $2
                ↪ 2>> temp.csv
        else
            time -f "%e" cp $1$(( 2 ** ($n) ))KB.bin $2
                ↪ 2>> temp.csv
        fi
    done
    rm $2*
    mv data.csv tempdata.csv
    paste -d, tempdata.csv temp.csv > data.csv
    rm temp.csv tempdata.csv
done
```

Listing 3: Verschiebung mit kleiner werdende Größen

4 Ergebnisse

Bei den Daten bei denen die Dateien nach absteigender Größe kopiert wurden traten einige interessante Ereignisse auf. Zunächst kostete der Vorgang mit der 2GB Datei mit dem FAT Filesystem als Quelle sowohl deutlich weniger Zeit als bei den anderen beiden als auch bei der 1GB Datei die darauf folgte. Im Schnitt dauerte der Vorgang von FAT zu F2FS 3,14 Sekunden, was 22,04 Sekunden schneller war als das von der 1GB großen Datei. Mit der gleichen Datei dauerte die Umkehrrichtung 76,93 Sekunden. Was dieses Vorkommen noch mehr als Ausreißer hervorruft ist, dass in dieser Kombination (von FAT zu F2FS und die Rückrichtung) die restlichen Daten sonst sehr ähnlich sind.

Die Performance-Vorteile die FAT als Quelle spiegelten sich jedoch nicht als Empfänger. Obwohl FAT im Schnitt 40,4 Sekunden schneller von sich aus kopieren lies als F2FS, war es als Ziel 87,46 Sekunden langsamer als F2FS wenn man die Zeit um alle Dateien zu kopieren betrachtet.

In dieser Reihenfolge waren die Ziele entscheidend in der Performance und die Daten der verschiedenen Quellsysteme für ein gemeinsames Ziel weichten nicht signifikant von einander ab. Die größte solcher Abweichungen war von F2FS mit ungefähr 8 Sekunden.

In der umgekehrten Reihenfolge, in der die kleinsten Dateien zuerst Kopiert wurden, wurden die Ergebnisse grob gespiegelt, jedoch zeigte sich deutlich mehr Varianz zwischen den Ergebnissen der Empfänger-Filesysteme.

Dass die ext4 Partition, die auf dem eindeutig schnelleren Speichermedium, als Empfänger von Daten sich mit Abstand am schnellsten war und als Sender keine Performance Vorteile zeigte, deutet darauf hin, dass das Empfänger Filesystem tatsächlich der entscheidende Faktor in der gesamten Performance war. Dass heißt, es ist nicht davon auszugehen, dass FAT als Filesystem besser als Quelle eines Kopiervorgangs ist, sondern dass ext4 und F2FS deutlich performanter sind im Schreiben der Daten.

Trotzdem kann man aus den Daten extrapolieren, dass FAT eine bessere Performance hat wenn die Partition beginnt voller zu werden und große Daten geschrieben müssen. Bei den steigenden Dateigrößen war FAT im schnitt 15 Sekunden schneller im Vergleich mit sich selber in der anderen Reihenfolge. F2FS war im gleichen Vergleich ungefähr 13 Sekunden langsamer. Trotzdem war aber F2FS in dem aufsteigenden Test fast doppelt so schnell wie FAT.

5 Fazit