Robinhood: Evaluation der Möglichkeiten und der Leistung

— Projekt Parallelrechnerevaluation —

Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

Vorgelegt von:	Julian Tobergte
E-Mail-Adresse:	2tobergt@informatik.uni-hamburg.de
Matrikelnummer:	6414935
Studiengang:	Informatik
Betreuer:	Julian Kunkel , Micheal Kuhn

Hamburg, den 22.04.2015

Abstract

In diesem Bericht werde ich darlegen, inwiefern ich die Evaluation der Leistung und der Möglichkeiten der Policy Engine Robinhood konkret durchgeführt habe.

Inhaltlich wird es dabei sowohl um eine geeignete Konfigurierung des Robinhood Service, als auch um die Handhabung der zugrundeliegenden Infrastruktur des Lustre Dateisystems und speziell der von Robinhood genutzten MySQL-Datenbank gehen.

Zusätzlich beurteile ich den Grad der Anwendbarkeit der verschiedenen Funktionen von Robinhood im Szenario der Benutzung auf dem Rechencluster des WR-Fachbereichs.

Ich komme dabei zu dem Ergebnis, dass die Anforderungen an die Leistung von Robinhood mit dem Stand der verfügbaren Software und Infrastruktur noch nicht erfüllt werden, es aber durch Erscheinen zukünftig anstehender Softwareupdates durchaus plausibel wäre und infolgedessen eine Benutzung durchaus von wirtschaftlichem und wissenschaftlichem Interesse ist.

Inhaltsverzeichnis

1. Einleitung

Zur Motivation ...

Die Robinhood Policy Engine ist ein vielfältiges Tool zum Verwalten von Inhalten von großen Dateisystemen. Viele der genutzten Funktionen eines Dateisystems leiden unter den riesigen Dateimengen eines Rechenclusters und werden inperformant bis unnutzbar, beispielsweise ein "find" Aufruf.

Dies ist deshalb sinnvoll, weil in den allermeißten heutig genutzten Dateisystemen die Metadaten der gespeicherten Objekte nicht zentral am selben Speicherbereich auf der Festplatte gespeichert werden. Stattdessen werden diese häufig, wie auch beispielsweise bei Ext4, nicht einzeln, sondern in Blockgruppen/Superblöcken zusammengefasst und über die Festplatte verteilt gespeichert. [ext4-disk-layout]

Daraus folgt, dass der Lesekopf beim Abfragen der Metainformationen vieler Dateien viele Male zum nächsten Informationsraum springen muss. Diese Tatsache bewirkt, dass weiträumige Abfragen wie etwa "find" langsam sind.

Aus diesem Grund ist es geradezu notwendig als Admin eines solchen System Mittel und Wege zu finden diese Probleme zu umgehen. Die Funktionsweise von Robinhood ist es die Metadaten der Dateien in einer eigenen Datenbank zu speichern und auf dieser die Massen-operationen auszuführen.

Die Engine ist hochgradig parallelisiert programmiert und damit speziell auf "High Performance Computing" Systeme ausgelegt, die mehrere Millionen Einträge besitzen und in der Größenordnung von Petabytes speichern. [1]

Die Hauptfunktionen von Robinhood sind:

- Benutzer- Lasterfassung und Verwaltung
- Schnellere "find" und "du" Implementationen
- Speziell anpassbare Benachrichtigungen über Dateisystemnutzung
- enge Zusammenarbeit mit Lustre Dateisystem
- Automatisierbare Datei Policies
- Dateisystem Recovery bei Katastrophen

[1] Paraphrasiert aus dem Robinhood Wiki[rbh-wiki]

2. Aufgabenstellung

Die Aufgabenstellung ist dabei die folgende:

Die Möglichkeiten von Robinhood sind unter dem Fokus der Benutzung auf dem Rechencluster des WR-Arbeitsbereiches zu evaluieren. Dabei soll die Leistung des Tools in Hinblick auf die aktuellen Anforderungen analysiert werden und die möglichen Optionen der Optimierung in Betracht gezogen werden.

Konkret ist die Größe und das Wachstum der Datenbank zu beachten, die nicht über die einstellige Gigabyte Größe wachsen soll. Desweiteren soll Robinhood im Laufenden Betrieb die hohen Dateioperationslasten von bis zu 80.000 Ops/s verarbeiten können und das ganze System möglichst wenig belasten. [1]

Zusätzlich sollen einfache Purge Policies auf Funktionalität getestet werden. Das ganze soll über die in Robinhood bereitgestellte Funktion die Lustre MDT Changelogs zu lesen laufen.

[1] Last kann in der Praxis auf 4 Bereiche mit je eigener Instanz aufgeteilt werden.

3. Design & Implementierung

Nach der Installation und der initialen Ausführung des Robinhood Daemons läuft der Entwicklungsprozess in einem stetigen Kreislauf aus Änderung und Evaluation. Neben der Optimierung der Konfiguration von Robinhood ist auch die Optimierung der darunterliegenden MySQL Architektur zu beachten. Dazu gehören die Mysqld-Konfiguration und die Wahl der Storage-Engine für eben jene. All diese Optimierungsprozesse können weitaus nebenläufig betrachtet werden, denn von negativen Seiteneffekten ist nicht auszugehen. Zusätzlich sind neu erschienene Versionen der verwendeten Software zu beachten und gegebenenfalls einzupflegen. Beispielsweise ist im Laufe des Projekts ein Versions Update von Robinhood von Version 2.5.3 auf 2.5.4 erschienen. [1]

3.1. Installation

Die Installation des Tools läuft über mehrere Schritte und beinhaltet zunächst die Benutzung des mitgelieferten "configure" Skripts welches das Makefile erzeugt. Dazu sind auf dem Cluster einige spezielle C-Flags nötig und weitere Argumente für das configure Skript neben dem obligatorischen "with-purpose=TMPFS". (siehe Listing 3.1).

Listing 3.1: Compile String

```
1 CFLAGS="-I/opt/lustre-server-src/lustre-release/libcfs/include/
    ↪ -I/opt/lustre-server-src/lustre-release/lustre/include/
    ↪ -I/opt/lustre-server-src/lustre-release/lnet/include/"
    ↪ ./configure
    ↪ --with-lustre=/opt/lustre-server-src/lustre-release/
    ↪ --with-purpose=TMPFS --enable-llapi-fork-support
    ↪ --enable-mds-stat --prefix=$HOME/pre/install/
```

Zusätzlich braucht Robinhood eine MySQL Datenbank zum speichern seiner Metadaten. Für die Erzeugung dieser wird ein eigenes Tool mitgeliefert.

Listing 3.2: Robinhood Datenbankerzeugungsskript

```
1 rbh-config create_db
```

Dieses erkennt auf dem Cluster jedoch nicht alle relevanten MySQL Services (mysqladmin, mysqld) und bricht ab. Deswegen wurde der Weg der manuellen Erzeugung gewählt. (siehe Listing 3.3)

[1] Siehe Robinhood Changelog [rbh-changelog]

Listing 3.3: Manuelle Datenbankerzeugung

```
mysqladmin create <robinhood_db_name>
mysql <robinhood_db_name>
    create user robinhood identified by password;
    GRANT USAGE ON robinhood_db_name.* TO 'robinhood'@'% ;
    GRANT ALL PRIVILEGES ON robinhood_db_name.* TO
       ↪ 'robinhood'@'%;
    exit;
mysql --user=robinhood --password=password --host=db_host
   ↪ robinhood_db_name
```

Danach ist die Datenbank erzeugt, aber nur ein leeres Datenbankschema vorhanden. Dieses wird beim ersten Start vom Robinhood Daemon automatisch angelegt.

3.2. Basiskonfiguration

Die aus dem Robinhood Wiki vorgegebene "very basic configuration" wurde auf die vorhandene Infrastruktur angepasst. (siehe Listing 3.4 (gekürzte Version))

Essentiell sind die Angaben des Dateisystem Pfad und Typ in "General", die Spezifikation des MySQL Servers in "ListManager", sowie des Lustre MDT in "ChangeLog".

Listing 3.4: Robinhood Basis Config Datei

```
General {
        fs_path = "/mnt/lustre";
        fs_type = lustre;
}
Log {
        log_file = "/home/tobergte/pre/log/tmp_fs.log";
        stats_interval = 60s;
}
ListManager {
    commit_behavior = autocommit;
        MySQL {
                server = west5;
                db = rbhdb;
                user = robinhood;
                password = ****;
        }
}
ChangeLog {
   MDT {
        mdt_name = "MDT0000";
```

```
21          reader_id = "cl1";
22      }
23          force_polling = ON;
24          polling_interval = 1s;
25  }
26
27  EntryProcessor {
28          max_batch_size = 1000;
29
30  }
```

3.3. Datenbankgröße

Das Wachstum der Datenbank muss erfasst werden um Prognosen für den Speicherplatz Verbrauch einer in Zukunft sehr großen Datenbank machen zu können. Dazu lassen sich sehr einfach Stichproben im kleinen Maßstab machen, welche über MySQL Queries realisiert werden.

Listing 3.5: dbcountquery

```
1  SELECT COUNT(*) FROM ENTRIES;
```

Listing 3.6: dbsizequery

```
1  SELECT table_schema
       ↪ "DB Name",
2      Round(Sum(data_length + index_length) / 1024 / 1024, 1)
          ↪ "DB Size in MB"
3  FROM   information_schema.tables
4  GROUP  BY table_schema;
```

Die gemessenen Daten sind in folgender Tabelle protokolliert.

Anzahl der Dateien	Größe der Datenbank in MB
600	13.1
10600	34.4
20600	59.5
30600	84.6

Tabelle 3.1.: Messung Größe der Datenbank

Aus den Daten lässt sich folgende Extrapolation erzeugen.

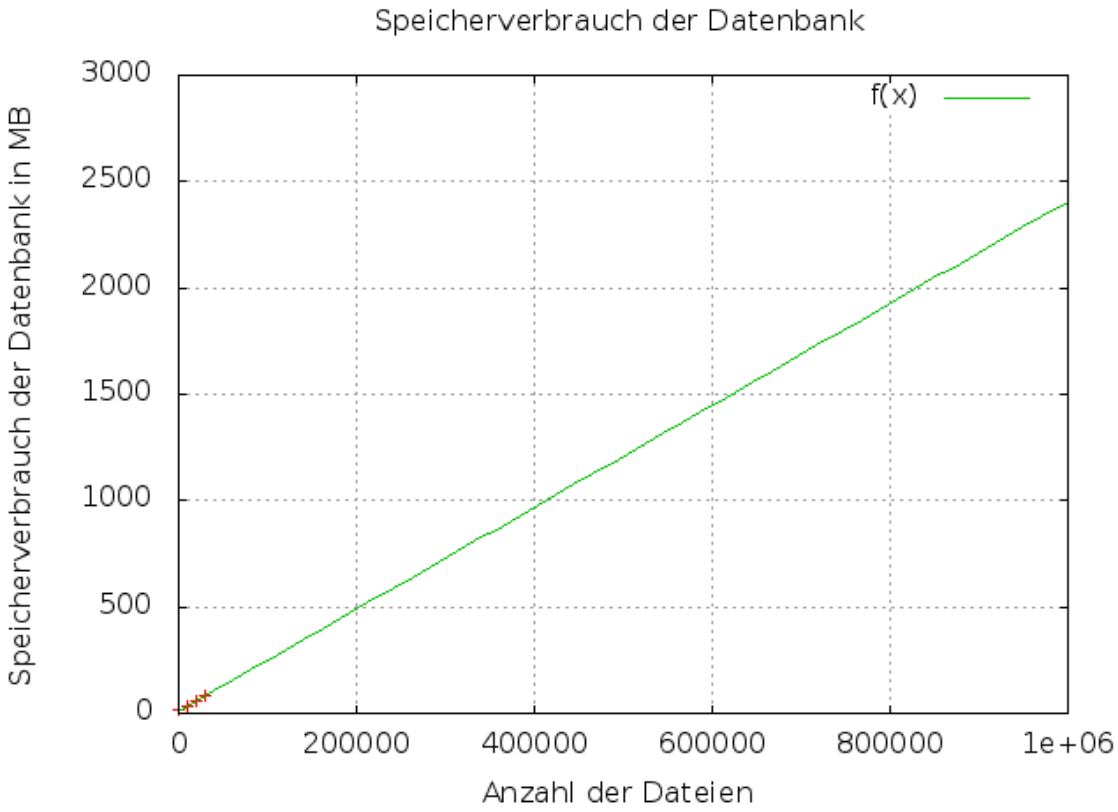

Abbildung 3.1.: Tabellendaten mit Extrapolation

Aus dem Graph lässt sich abschätzen, dass das Wachstum der Datenbank linear erfolgt. Dieses wird durch die kleinen gemessenen Samples und durch Angaben der Entwickler bestätigt. Als Richtlinie der Größe ist 1 Gigabyte pro 2.5 Millionen Einträge angemessen. Diese Größe liegt im akzeptablen Bereich für die Anwendung auf dem DKRZ Rechencluster.

3.4. Optimierung RBH-Daemon

Für die Optimierung der RobinHood-Konfiguration sind gleichbleibende Testbedingungen notwendig. Dazu gehört ein Benchmark-Skript und das Löschen des Cache vor jedem Test mit ”sudo /home/hr/drop-caches.sh” .

Listing 3.7: Benchmark-Skript ohne ”echo”-Statements

```
1  numfiles=100000
2  date
3  ./getdbcount.sh
4  ./getdbsize.sh
5  ./parabench/parabench ./parabench/create.pbl
     ↪ env=/mnt/lustre/tobergte/tmp0 create=$numfiles dirs=1
6  date
7  (/usr/bin/time sudo /var/robinhood --readlog=0 --once)
     ↪ 2>>benchmark_tmp.txt
8  ./getdbcount.sh
9  ./getdbsize.sh
10 ./parabench/parabench ./parabench/create.pbl
     ↪ env=/mnt/lustre/tobergte/tmp1 create=$numfiles dirs=1
11 date
12 (/usr/bin/time sudo /var/robinhood --readlog=0 --once)
     ↪ 2>>benchmark_tmp.txt
13 ./getdbcount.sh
14 ./getdbsize.sh
15 ./parabench/parabench ./parabench/create.pbl
     ↪ env=/mnt/lustre/tobergte/tmp2 create=$numfiles dirs=1
16 date
17 (/usr/bin/time sudo /var/robinhood --readlog=0 --once)
     ↪ 2>>benchmark_tmp.txt
18 ./getdbcount.sh
19 ./getdbsize.sh
```

Dieses benutzt Parabench um geregelt Test-Dateien zu erzeugen. Es werden 3 mal je ”numfiles” Dateien erzeugt und jedes mal davor die Anzahl der Datenbankeinträge ausgegeben um die Korrektheit der Ausführung überprüfen zu können. Hauptfunktion ist, dass mit Hilfe von ”/usr/bin/time” die Zeit gemessen wird, die von RobinHood gebraucht wird um die aufgetretenen Dateiänderungen zu verarbeiten und in die Datenbank zu schreiben. Diese Zeitangabe wird in eine Textdatei geschrieben. Anzumerken ist, dass hierbei nur die Zeit für Insert-Operationen gemessen wird, welche schneller sind als die Delete-Operationen, jedoch langsamer als Update-Operationen.

Als Gegenstück dazu gibt es das Clear-Skript, welches dafür sorgt, dass jeder Benchmark-Lauf mit der selben kleinen Datenbank startet. Dieses kann auch zur Messung der Zeit für Delete-Operationen verwendet werden (siehe Tabelle 3.6).

Listing 3.8: Clear-Skript ohne ”echo”-Statements

```
1 rm -r /mnt/lustre/tobergte/tmp0
2 rm -r /mnt/lustre/tobergte/tmp1
3 rm -r /mnt/lustre/tobergte/tmp2
```

```
4 (/usr/bin/time sudo /var/robinhood --readlog=0 --once)
    ↪ 2>>benchmark_clear_tmp.txt
```

In der folgenden Tabelle sind ansatzweise die verschiedenen sich positiv auswirkenden Testläufe protokolliert. Selbstverständlich ist dabei der Umfang eines vollständigen Tests aller möglichen Permutationen von Konfigurationen exponentiell wachsend und im Fall von Robinhood nicht komplett ausführbar. Stattdessen wurde durch "schlaues Raten" versucht ein möglichst gutes Ergebnis zu erreichen. Diese sind examplarisch in der folgenden Tabelle festgehalten.

Für eine Aufschlüsselung der Konfig-Bereiche siehe Basiskonfiguration 3.4 .

Parameter	Wert	Avg Zeit
EP::max batchsize	100	0:50.33
EP::max batchsize	1000	0:09.13
CL::max queuesize	1	0:09.13
CL::max queuesize	100	0:08.87

Tabelle 3.2.: Messung RBH bei 10.000 Dateien

Als Ergebnis zeigt sich, dass sich einige schlecht gewählte Parameter sehr viel stärker negativ auswirken als andere, die sich bei falscher Ausprägung wenig stark negativ auf die Performance auswirken. Dies war zu erwarten. Daraus folgt als Konsequenz, dass es sich nicht lohnt jeden Parameter auf Optimierungsbedarf zu untersuchen. Die gemessene Zeit ließ sich um einiges vom Ausgangswert verbessern, jedoch ist abzusehen, dass es weitere Optimierungsmöglichkeiten gibt, dessen Ausmaß aber wahrscheinlich recht gering ausfallen würde, ausgehend vom bereits erreichten Speedup. Weitere Tests wurden dann mit einem größeren Ausmaß von 3 mal 100.000 Dateien durchgeführt, um längere Laufzeiten als wenige Sekunden zu erhalten. Der Grund liegt darin, dass bei einer Laufzeit von etwa 1 Minute die Äußeren Einflüsse minimiert werden, bei annehmbarer Ausführungszeit. Diese war durch die vorangehenden Optimierungen nämlich nicht mehr gegeben. Weitere erhebliche Hebelpunkte bietet die Datenbank, dessen Optimierung im nächsten Kapitel behandelt wird.

3.5. Optimierung MySQLd

Der Vorteil der Optimierungen des MySQL-Daemon ist, dass eine weitaus größere Nutzerbasis vorhanden ist, welche ebenfalls versucht die beste Konfiguration für jeden Anwendungsfall zu finden. Diese Tatsache ist natürlich auch gleichzeitig ein Nachteil oder Risiko, denn die schiere Menge an Ergebnissen bei einer Suche, sorgt für einen immensen Filteraufwand für den Erhalt von sinnvollen Ratschlägen.

3.5.1. Konfiguration MySQLd

Auf der Wiki-Seite wird dazu geraten folgende MySQLd-Konfig-Parameter zu schreiben.

Listing 3.9: mysql-cnf

```
performance_schema
table_open_cache=2028
max_connections=1024
thread_cache_size=512
table_cache=2048
connect_timeout=60
key_buffer_size=512M
query_cache_size=512M
query_cache_limit=512M
sort_buffer_size=512M
read_rnd_buffer_size=1G
tmp_table_size=1G
max_heap_table_size=1G
```

Diese bringen jedoch keinen Geschwindigkeitsbonus, eher noch einen Abfall.

Lauf	Avg Zeit
normal	1:29.62
mit mysql opt 3.9	1:32.43

Tabelle 3.3.: Vergleich MySQL-Optimierung bei 100.000 Dateien

Außerdem wird dazu geraten folgende Innodb-Optimierungen vorzunehmen.

Listing 3.10: mysql-cnf-innodb

```
# Fine Tuning (for rbh)
innodb_file_per_table
# 50% to 90% of the physical memory
innodb_buffer_pool_size=4G
# 2*nbr_cpu_cores
innodb_thread_concurrency=12
# memory cache tuning
innodb_max_dirty_pages_pct=15
# robinhood is massively multithreaded: set enough
   ↪ connections for its threads, and its multiple instances
max_connections=256
# increase this parameter if you get DB connection failures
connect_timeout=60
# This parameter appears to have a significant impact on
   ↪ performances:
```

```
14 # see the following article to tune it appropriately:
15 #
     ↪ http://www.mysqlperformanceblog.com/2008/11/21/how-to-calculate
16 innodb_log_file_size=500M
17 innodb_log_buffer_size=8M
18
19 # proly good for removing entries
20 # seems to be bad
21 #key_buffer_size=512M
```

Die Optimierungen für InnoDB hingegen führen zu sehr guten Ergebnissen. Die gemessene Zeit liegt auch bei der kleinen Testgröße deutlich unter der Vergleichsmessung.

Lauf / Zeit	Avg Zeit
normal	0:08.52
mit innodb opt 3.10	0:06.83

Tabelle 3.4.: Vergleich InnoDB-Optimierung bei 10.000 Dateien

3.5.2. Wahl der Storage-Engine

Standardmässig wird als Storage-Engine für MySQL Innodb verwendet. Diese steht für hohe Konsistenz der Transaktionen und geringes Risiko für Datenbankkorruptionen. Allerdings sind DELETE Statements dafür auch sehr langsam. Eine Alternative dazu bietet MyISAM.

Auf die Frage nach den Nachteilen der Benutzung der MyISAM Storage-Engine antwortete Thomas Leibovici, einer der Hauptentwickler von Robinhood, auf der Support Mailingliste:

"There is no problem using MyISAM, except its transaction management is weaker, so you have a higher risk of corrupting the DB or creating inconsistencies in case of DB host crash." [rbh-support-myisam]

Deshalb sollen die verfügbaren Storage-Engine in diesem Kapitel auf ihre Leistung hin geprüft werden. Aktuell sind jedoch nur InnoDB und MyISAM unterstützt, für weitere Informationen siehe Fazit.

Durchführung

Um die Storage-Engine zu wechseln muss Robinhood beendet und der "engine" Parameter im MySQL Bereich der Robinhood-config angepasst werden. [2]

[2] seit v.2.5.4, früher innodb=false [rbh-changelog]

Danach muss entweder die Datenbank gelöscht und neu angelegt werden oder man konvertiert die Datenbank per SQL Statement von Hand. Das neu Anlegen ist aber einfacher und schneller.

Listing 3.11: engine-parameter

```
1  ListManager {
2      .
3      .
4          MySQL {
5                  .
6                  .
7                  #engine = innodb;
8                  engine = myisam;
9          }
10 }
11 ---------
12 ALTER TABLE t1 ENGINE=xxxx;
```

Evaluation

Für die Vergleichswerte wurden wieder 3 mal je 100.000 Dateien erzeugt und die Insert-Zeit gemessen. Diese Fallen ähnlich aber doch zum erkenntlichen Vorteil der InnoDB-Engine aus.

Lauf	Lauf 1	Lauf 2	Lauf 3	Avg	Ops/s
innodb	1:02	1:08	1:12	1:07	1492
myisam	1:28	1:38	1:21	1:27	1149

Tabelle 3.5.: Messung 100.000 Insert Operationen

Allerdings sind die Lösch-operationen bei der Innodb-Engine bedeutend langsamer und schon auf der relativ kleinen Testgrösse geradezu unbenutzbar mit Laufzeiten von knapp unter einer Stunde. Unter myisam kann die Datenbank immernoch nicht mit den Einfüge-operationen mithalten, sie ist jedoch nur etwa um den Faktor 0.5 langsamer.

Fazit ist, dass die Einbußen der myisam-Engine bei den Insert-Operationen (-25%) zu verkraften sind, wenn dafür die weitaus schnelleren (+2035%) Delete-Operationen verwendet werden können. Die Angesprochenen möglichen Inkonsistenzen sind leider

Lauf	Lauf 1	Lauf 2	Lauf 3	Avg	Ops/s
innodb	51:35	54:58	53:29	53:20	31
myisam	2:32	2:24	2:37	2:31	662

Tabelle 3.6.: Messung 100.000 Delete Operationen

schwer bis überhaupt nicht objektiv zu bewerten, da sie ausschlieslich im Falle des Server-Absturzes auftreten. Solch ein Fall ist dann zwar unschön, aber kein großes Problem, da die Datenbank relativ einfach neu angelegt werden kann, weil in ihr nur redundate Informationen gespeichert werden.

Aus diesen Gründen fällt die Wahl auf MyISAM als bessere Storage-Engine.

Für die Robinhood-Version v3.0, dessen Verofffentlichung im 3. bzw. 4. Quartal 2015 angesetzt ist, ist die Unterstützung von anderen Storage-Engine wie z.B. PGSQL geplant. [rbh-control]

3.5.3. Fazit

Der MySQL-Daemon lässt sich recht leicht anpassen und Optimierungen können einfach getestet werden. Diese sind ausdrücklich notwendig und verschaffen vieles an zusätzlichem Leistungsgewinn.

Die InnoDB-Storage-Engine lässt sich zwar weitreichend konfigurieren um mehr Leistung zu erhalten, aber MyISAM als Alternative bietet letztendlich den größeren Vorteil.

Mit Robinhood v3.0 müssten die zusätzlich unterstützten Storage-Engine auf die Probe gestellt werden.

Die neueste Version von MySQL 5.6 verspricht Speedups von 150-230% beim Schreiben bzw. Lesen. [mysql-5.6-bench]

Nicht außer Acht zu lassen ist natürlich auch die Cluster-Version von MySQL, welche linear mit der Anzahl der benutzten Knoten skaliert und somit wiederrum ein Vielfaches der Leistung ermöglicht. Das Aufsetzen einer solchen Infrastruktur würde sich also durchaus lohnen, da die Einfüge-Operationen in die Datenbank das Bottleneck darstellen. [rbh-support-myisam]

3.6. Evaluation der Policy Anwendung

In Robinhood v2.5.x gibt es eine begrenzte Menge an Policies, die über den Aufruf des ./configure-Skripts zur Compile-Zeit statisch definiert werden. Im Standardfall von "robinhood-tmpfs" (siehe Kapitel 3.1) sind "purge policy" und "rmdir policy" verfügbar. Für das gesamte Sortiment im Überblick siehe Tabelle 3.7.

Package / Policy	”migration”	”purge”	”hsm remove”	”rmdir”
-tmpfs	-	rm (old files)	-	rmdir, rm -rf
-backup	Copy to storage backend	-	rm in storage backend	-
-lhsm	Lustre HSM archive	Lustre HSM release	Lustre HSM remove	-

Tabelle 3.7.: Policy-Verfügbarkeit pro Package [rbh-control]

In Robinhood v3.0 soll es generische Policies geben, die alle im Standard Package benutzt werden können, ohne extra eine ”with-purpose” Flag an das configure-Skript zu übergeben.

4. Fazit

Das Fazit der Evaluation der Möglichkeitung und der Leistung ist recht gefächert. Zunächst erweist sich die befürchtete Speicherhungrigkeit der Datenbank mit einem Ausmaß von etwa 2.5GB pro Millionen Einträge als nichtig. Die Benutzung von Robinhood allgemein gestaltet sich vielfältig von spannend über verblüffend bis hin zu anstrengend. Letzteres eben genau auf Grund der Vielfalt des Programms die nicht weitreichend genug im Benutzerhandbuch und in den zahlreichen Dokumentationen dargestellt werden kann.

Bei der Wartung und Optimierung kommen häufig die Fragen auf "Ist diese Rate hoch genug?", "Ist da noch mehr möglich?" und sie lassen sich nicht beantworten, denn es gibt einerseits zu wenig Vergleichswerte, eben weil die Verbreitung der Benutzer die Robinhood benutzen sehr gering ist und der Anteil der Benutzer, die ihre Leistungsdaten veröffentlichen entsprechend geringer. Andererseits existiert das Problem des Vergleichs von Leistungsdaten grundsätzlich, denn jeder Cluster auf dem gerechnet wird hat seine eigene Hardwarezusammenstellung und eigene Menge an Knoten und Prozessoren. Zu dem kommen noch die unterschiedlichen Jobverwaltungs- und Dateisysteme, deren Versionspermutationen zahlreich sind. Aus diesen genannten Gründen ist ein Vergleich von Performance mit anderen Gleichgesinnten nicht zielführend und von geringem Informationsgehalt. Es ist natürlich trotzdem immer möglich die eigene Performance mit den Musterangaben aus der Dokumentation und dem Wiki zu vergleichen und daraus Rückschlüsse zu ziehen und nach dem Erreichen dieses Grenzwertes zu streben.

Grundsätzlich kann man für die Leistung von Robinhood sagen, dass es neben den schon erwähnten Stellschrauben selbstverständlich noch weitere in der Konfigurationsdatei gibt, dessen Potential noch nicht ausgeschöpft wurde. Der größte Hebel aber, an dem man ansetzen sollte, ist die dahinter liegende Datenbank. Begründen kann man dies damit, dass die Datenbank das "Bottleneck" im Programmfluss darstellt. Die genauere Betrachtung jener brachte mit Abstand den effizientesten Speedup und dazu noch die Aussicht auf mehr, anlässlich der neuen MySQL-Versionen und der kommenden Unterstützung von anderen Storage-Engine. Darüber hinaus ist die Möglichkeit der Benutzung der Cluster-Version von MySQL allgegenwärtig, welche mit der Anzahl der verwendeten Knoten skaliert.

Für den Betrachtungswinkel der Benutzung auf dem WR-Cluster ist die aktuelle Leistung noch nicht den Anforderungen entsprechend. Es bedarf den Faktor 10 für die Performance. Dieser ist mit aktuellen Mitteln und Möglichkeiten nicht zu erreichen, jedoch kann dies im Laufe der Zeit der Fall sein, sobald die neuen Versionen der entsprechenden Software verfügbar sind. In dem Fall wäre es tatsächlich wirtschaftlich die richtige Entscheidung Robinhood aktiv zu benutzen, aufgrund der genannten Zeit- und Arbeitsersparnis, wissenschaftlich gesehen ist es natürlich schon jetzt eine gute Entscheidung.

Literaturverzeichnis

[ext4-disk-layout] Ext4 Dev. Ext4 disk layout. https://ext4.wiki.kernel.org/index.php/Ext4_Disk_Layout, 2015. [Online; accessed 22-April-2015].

[myisam] MySQL. The myisam storage engine. http://dev.mysql.com/doc/refman/5.0/en/myisam-storage-engine.html, 2015. [Online; accessed 22-April-2015].

[mysql-5.6] MySQL. Dba and developer guide to mysql 5.6. http://dev.mysql.com/tech-resources/articles/mysql-5.6.html, 2015. [Online; accessed 22-April-2015].

[mysql-5.6-bench] Oracle. Mysql 5.6 benchmarks. http://www.mysql.de/why-mysql/benchmarks/, 2015. [Online; accessed 22-April-2015].

[mysql-doc] MySQL. 5.5 documentation. http://dev.mysql.com/doc/refman/5.5/en/glossary.html, 2015. [Online; accessed 22-April-2015].

[rbh-changelog] Robinhood. Changelog. https://github.com/cea-hpc/robinhood/wiki/ChangeLog, 2015. [Online; accessed 22-April-2015].

[rbh-control] Thomas Leibovici. Taking back control of hpc file systems with robinhood policy engine. http://lustre.ornl.gov/ecosystem/documents/LustreEco2015-Leibovici.pdf, 2015. [Online; accessed 22-April-2015].

[rbh-support] Robinhood. Support mailinglist. http://sourceforge.net/p/robinhood/mailman/robinhood-support/, 2015. [Online; accessed 22-April-2015].

[rbh-support-myisam] Thomas Leibovici. [robinhood-support] going back to myisam. http://sourceforge.net/p/robinhood/mailman/message/31847461/, 2015. [Online; accessed 22-April-2015].

[rbh-wiki] Robinhood. Wiki. https://github.com/cea-hpc/robinhood/wiki, 2015. [Online; accessed 22-April-2015].

[redo-logs] Annamalai Gurusami. Redo logging in innodb. https://blogs.oracle.com/mysqlinnodb/entry/redo_logging_in_innodb, 2015. [Online; accessed 22-April-2015].

[tmpfs-adminguide] Robinhood. tmpfs admin guide. https://github.com/cea-hpc/robinhood/wiki/tmpfs_admin_guide, 2015. [Online; accessed 22-April-2015].

[tmpfs-tutorial] Robinhood. tmpfs tutorial. https://github.com/cea-hpc/robinhood/wiki/tmpfs_tutorial, 2015. [Online; accessed 22-April-2015].

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

Anhänge

A. Anhangskapitel

Genutzte Software/Hardware während des Projekts

Zur Berichterstellung

- latex+bibtex mit der bereitgestellten Vorlage
- gnuplot

Zur Projektarbeit

- robinhood
- ssh-session des WR-Cluster und alle seine implizierten Programme (slurm, ..)
- parabench für Testdaten
- mysqld für die Datenbank