1 Überblick

1.1 Einführung in Benchmarks

Benchmarks dienen dazu, praktisch und effektiv zu untersuchen, wie sich ein System unter Last verhält. Die wichtigste Erkenntnis, die man aus Benchmarks gewinnen kann, sind die Probleme und Fehler, die man systematisch dokumentieren und nach Priorität abarbeiten sollte. Das Ziel von Benchmarks ist die Reduzierung und Bewertung von unerwünschtem Verhalten sowie die Analyse, wie sich das System derzeit und unter simulierten, zukünftigen, anspruchsvolleren Bedingungen verhalten könnte.

Es gibt zwei verschiedene Techniken für Benchmarks. Die erste zielt darauf ab, die Applikation als Ganzes zu testen (full-stack). Dabei wird nicht nur die Datenbank getestet, sondern die gesamte Applikation, einschließlich des Webservers, des Netzwerks und des Applikationscodes. Der Ansatz dahinter ist, dass ein Nutzer genauso lange auf eine Abfrage warten muss, wie das gesamte System benötigt. Daher sollte diese Wartezeit so gering wie möglich sein. Es kann dabei vorkommen, dass MySQL nicht immer das Bottleneck ist.[1]

Full-Stack-Benchmarks haben jedoch auch Nachteile. Sie sind schwieriger zu erstellen und insbesondere schwieriger korrekt einzurichten. Wenn man lediglich verschiedene Schemas und Abfragen in MySQL auf ihre Performance testen möchte, gibt es sogenannte Single-Component-Benchmarks. Diese analysieren ein spezifisches Problem in der Applikation und sind deutlich einfacher zu erstellen. Ein weiterer Vorteil besteht darin, dass nur ein Teil des gesamten Systems getestet wird, wodurch die Antwortzeiten kürzer sind und man schneller Ergebnisse erhält.

Wenn bei Benchmarks schlechte Designentscheidungen getroffen werden, kann dies zu einer falschen Interpretation des Systems führen, da die Ergebnisse nicht die Realität widerspiegeln. Die Größe des Datensatzes und des Workloads muss realistisch sein. Idealerweise verwendet

[1] Gemeint ist ein Engpass beim Transport von Daten oder Waren, der maßgeblichen Einfluss auf die Abarbeitungsgeschwindigkeit hat. Optimierungsversuche an anderer Stelle führen oft nur zu geringen oder gar keinen messbaren Verbesserungen der Gesamtsituation. (https://martinvogel.de/lexikon/bottleneck.html)

man einen Snapshot[2] des tatsächlichen produktiven Datensatzes. Gibt es keine Produktionsdaten, sollten die Daten und der Workload simuliert werden, da realistische Benchmarks komplex und zeitaufwendig sein können.

Häufige Fehler beim Durchführen von Benchmarks sind unter anderem, dass nur ein kleiner Teil der tatsächlichen Datensatzgröße verwendet wird und die Datensätze unkorrekt gleichmäßig verteilt sind. In der Realität können Hotspots auftreten. Bei zufällig generierten Werten kommt es hingegen häufig zu unrealistisch gleichmäßig verteilten Datensätzen. Ein weiterer Fehler besteht darin, dass man beim Testen einer Anwendung nicht das tatsächliche Benutzerverhalten nachstellt. Wenn gleiche Abfragen in einer Schleife ausgeführt werden, muss man außerdem auf das Caching achten, da sonst falsche Annahmen über die Performance getroffen werden können. Zudem wird oft die Warmmachphase des Systems vollständig ignoriert. Kurze Benchmarks können schnell zu falschen Annahmen über die Performance des Systems führen.

Um verlässliche Ergebnisse zu erhalten, sollte ein Benchmark ausreichend lange laufen, um den stabilen Zustand des Systems zu beobachten, insbesondere bei Servern mit großen Datenmengen und viel Speicher. Dabei ist es wichtig, so viele Informationen wie möglich zu erfassen und sicherzustellen, dass der Benchmark wiederholbar ist, da unzureichende oder fehlerhafte Tests wertlos sind. Außerdem ist es wichtig, die Ergebnisse in einem Diagramm darzustellen, da auftretende Phänomene sonst anhand einer tabellarischen Darstellung nicht erkannt werden können.

1.2 Measures

- **Durchsatz (Throughput):** Der Durchsatz ist die Anzahl an Transaktionen pro Zeiteinheit. Er ist standardisiert, und Datenbankanbieter versuchen, diesen zu optimieren. Meistens werden Transaktionen pro Sekunde (oder manchmal pro Minute) als Einheit verwendet.
- **Antwortzeiten (Latenz):** Die Antwortzeit misst die gesamte Zeit, die für eine Abfrage benötigt wird. Diese kann, abhängig von der Applikation, in Mikrosekunden (µs), Millisekunden (ms), Sekunden oder Minuten angegeben werden. Von dieser Zeit können aggregierte Antwortzeiten wie Durchschnitt, Maximum, Minimum und Perzentile abgeleitet werden. Das Maximum ist oft eine weniger sinnvolle Metrik, da es sich nicht gut wiederholen lässt. Daher nutzt man eher Perzentile bei den Antwortzeiten. Wenn beispielsweise das 95. Perzentil der Antwortzeit bei 5 ms liegt, bedeutet dies, dass mit einer Wahrscheinlichkeit von 95 % die Abfrage in weniger als 5 ms abgeschlossen ist.

[2] Snapshots bestehen größtenteils aus Metadaten, die den Zustand Ihrer Daten definieren, und sind keine vollständige Duplikation der Daten auf Ihrer Festplatte. Snapshots werden häufig für Test-/Entwicklungsaufgaben verwendet. (https://www.rubrik.com/de/insights/what-is-a-snapshot-backup)

- **Nebenläufigkeit (Concurrency):** Die Nebenläufigkeit auf dem Webserver lässt sich nicht zwangsläufig auf den Datenbankserver übertragen. Eine genauere Messung der Gleichzeitigkeit auf dem Webserver besteht darin, zu bestimmen, wie viele gleichzeitige Anfragen zu einem bestimmten Zeitpunkt ausgeführt werden. Es kann auch geprüft werden, ob der Durchsatz sinkt oder die Antwortzeiten steigen, wenn die Gleichzeitigkeit zunimmt. Beispielsweise benötigt eine Website mit „50.000 Benutzern gleichzeitig“ vielleicht nur 10 oder 15 gleichzeitig laufende Abfragen.
- **Skalierbarkeit (Scalability):** Skalierbarkeit ist wichtig für Systeme, die ihre Performance unter unterschiedlich starken Workloads beibehalten müssen. Ein ideales System würde doppelt so viele Abfragen beantworten (Throughput), wenn doppelt so viele „Arbeiter“ versuchen, die Aufgaben zu erfüllen. Die meisten Systeme sind jedoch nicht linear skalierbar und zeigen Leistungseinbußen, wenn die Parameter variieren.

1.3 Tools

1.3.1 Einführung

- Webhoster Wissen - MySQL Benchmark mit Sysbench
- Sysbench GitHub Repository

Als Haupttool, um Benchmarktests durchzuführen, habe ich mich für Sysbench entschieden. Sysbench ist ein Open-Source-Tool, das ein skriptfähiges, multi-threaded Benchmark-Tool, das auf LuaJIT basiert. Es wird auch hauptsächlich für Datenbankbenchmarks verwendet, kann jedoch auch dazu eingesetzt werden, beliebig komplexe Arbeitslasten zu erstellen, die keinen Datenbankserver erfordern. Dabei werden Tests auf verschiedenen Systemressourcen, wie CPU, Speicher, I/O und Datenbanken wie MySQL verwendet.

Im Zuge der Recherchearbeit habe ich mir auch andere Benchmarking-Tools betrachtet, wie z.B. Benchbase (GitHub) oder mybench (GitHub). Die größten Vorteile von Sysbench habe ich in der Skriptfähigkeit und Flexibilität gesehen. D.h. dass ich benutzerdefinierte Benchmarks schneller und unkompliziert erstellen kann. Außerdem hat sich Sysbench als de facto Standard im Bereich der Datenbankbenchmarks etabliert (mybench). Dadurch stehen eine breite Nutzerbasis und viele verfügbare Ressourcen zur Verfügung. Im Vergleich zu den anderen Tools bietet allerdings Sysbench eine weniger präzise Steuerung der Ergebnisrate und der Transaktionen. Außerdem haben Tools wie mybench die Möglichkeit, in Echtzeit umfassende Visualisierungen darzustellen. Damit können Metriken live in einem Diagramm angezeigt werden (mybench Live Monitoring). Dieses Feature ist sicherlich hilfreich, aber in meinem Fall habe ich abgewogen und bin zu dem Entschluss gekommen, dass die einfachere

Bedienung für mich der ausschlaggebende Grund, neben dem Fakt, dass Sysbench der de facto Standard ist.

Da die Graphen aber trotzdem eine entscheidende Rolle bei der Analyse darstellen, werde ich das Tool Gnuplot (GitHub) dafür benutzen, um die Werte nach der Durchführung des Benchmarks zu visualisieren.

1.3.2 Kurze Einführung in die Tools

Zunächst müssen die beiden Tools installiert werden. Auf einem Mac erfolgt dies über:

```
brew install sysbench
brew install gnuplot
```

Mit der Hilfe von Homebrew. Des Weiteren muss der MySQL-Server korrekt gestartet sein und eine Datenbank erstellt werden mit:

```
CREATE DATABASE sbtest;
```

Um einfach Testdaten in die Datenbank einzufügen, kann dieser Befehl verwendet werden:

```
sysbench --db-driver=mysql --mysql-host=localhost --mysql-user=YOUR_USER --mysql-
    password=YOUR_PASSWORD --mysql-db=sbtest oltp_insert --tables=10 --table-size
    =100000 prepare
```

`YOUR_USER` und `YOUR_PASSWORD` müssen entsprechend ersetzt werden.

Codeblock 1.1: Sysbench Script

```
#!/bin/bash

# File Paths
OUTPUT_FILE="output/sysbench_output.csv"
RAW_RESULTS_FILE="output/sysbench.log"
GNUPLOT_SCRIPT="plot_sysbench.gp"

# Connection parameters
DB_HOST="localhost"
DB_USER="root"
DB_PASS="password"
DB_NAME="sbtest"
TABLES=10
```

```
TABLE_SIZE=10000
DURATION=10

echo "Preparing the database..."
sysbench oltp_read_write --db-driver=mysql --mysql-host=$DB_HOST --mysql-user=$DB_USER --mysql-password=$DB_PASS --mysql-db=$DB_NAME --tables=$TABLES --table-size=$TABLE_SIZE prepare > $RAW_RESULTS_FILE 2>&1
echo "Database prepared."

echo "Running benchmark..."
sysbench oltp_read_write --db-driver=mysql --mysql-host=$DB_HOST --mysql-user=$DB_USER --mysql-password=$DB_PASS --mysql-db=$DB_NAME --tables=$TABLES --table-size=$TABLE_SIZE --time=$DURATION --threads=1 --report-interval=1 run >> $RAW_RESULTS_FILE 2>&1

echo "Time (s),Threads,TPS,QPS,Reads,Writes,Other,Latency (ms,95%),Err/s,Reconn/s" > "$OUTPUT_FILE"

# Extract the relevant lines and format as CSV
grep '^\[ ' $RAW_RESULTS_FILE | while read -r line; do
    time=$(echo $line | awk '{print $2}' | sed 's/s//')

    threads=$(echo $line | awk -F 'thds: ' '{print $2}' | awk '{print $1}')
    tps=$(echo $line | awk -F 'tps: ' '{print $2}' | awk '{print $1}')
    qps=$(echo $line | awk -F 'qps: ' '{print $2}' | awk '{print $1}')

    read_write_other=$(echo $line | sed -E 's/.*\(r\/w\/o: ([0-9.]+)\/([0-9.]+)\/([0-9.]+)\).*/\1,\2,\3/')
    reads=$(echo $read_write_other | cut -d',' -f1)
    writes=$(echo $read_write_other | cut -d',' -f2)
    other=$(echo $read_write_other | cut -d',' -f3)

    latency=$(echo $line | awk -F 'lat \\(ms,95%\\): ' '{print $2}' | awk '{print $1}')
    err_per_sec=$(echo $line | awk -F 'err/s: ' '{print $2}' | awk '{print $1}')
    reconn_per_sec=$(echo $line | awk -F 'reconn/s: ' '{print $2}' | awk '{print $1}')

    echo "$time,$threads,$tps,$qps,$reads,$writes,$other,$latency,$err_per_sec,$reconn_per_sec" >> "$OUTPUT_FILE"
done

```

```
echo "Benchmark complete. Results saved to $OUTPUT_FILE."

# Generate the Gnuplot graph
echo "Generating plot..."
gnuplot $GNUPLOT_SCRIPT
echo "Plot generated."

echo "Cleaning up..."
sysbench oltp_read_write --db-driver=mysql --mysql-host=$DB_HOST --mysql-user=
    $DB_USER --mysql-password=$DB_PASS --mysql-db=$DB_NAME --tables=$TABLES cleanup
    >> $RAW_RESULTS_FILE 2>&1
echo "Database cleanup complete."
```

Codeblock 1.2: Gnuplot Script

```
set datafile separator ","
set title "Benchmark Results: TPS, Latency, Queries, and More"
set xlabel "Time (s)"
set ylabel "Values"
set grid
set key outside
set terminal pngcairo enhanced font 'Arial,10'
set output "/Users/danielmendes/Desktop/Bachelorarbeit/Ausarbeitung/Tools/Output/
    sysbench_output_plot.png"
set yrange [0:*]

# Plot each attribute on its own line
plot "/Users/danielmendes/Desktop/Bachelorarbeit/Ausarbeitung/Tools/Output/
    sysbench_output.csv" using 1:2 title "Threads" lt 1 lc rgb "black" with lines, \
     "" using 1:3 title "TPS" lt 2 lc rgb "green" with lines, \
     "" using 1:4 title "QPS" lt 3 lc rgb "blue" with lines, \
     "" using 1:5 title "Reads" lt 4 lc rgb "red" with lines, \
     "" using 1:6 title "Writes" lt 5 lc rgb "orange" with lines, \
     "" using 1:7 title "Other" lt 6 lc rgb "purple" with lines, \
     "" using 1:8 title "Latency (ms)" lt 7 lc rgb "cyan" with lines, \
     "" using 1:9 title "Err/s" lt 8 lc rgb "magenta" with lines, \
     "" using 1:10 title "Reconn/s" lt 9 lc rgb "brown" with lines
```