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. (Vogel, 2009)

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. (Germany, 2024)

- **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

Als Haupttool, um Benchmarktests durchzuführen, habe ich mich für Sysbench akopytov, 2024 entschieden. Sysbench ist ein Open-Source-Tool, das ein skriptfähiges, multi-threaded Benchmark-Tool ist, 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, S peicher, I/O und Datenbanken wie MySQL Reimers, 2017 verwendet.

Im Zuge der Recherchearbeit habe ich mir auch andere Benchmarking-Tools betrachtet, wie z.B. Benchbase Difallah et al., 2013 oder mybench Shopify, 2024. 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 Shopify, 2022b. 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 Shopify, 2022a. 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.

Trotzdem kann man nicht komplett auf Graphen verzichten, da beispielweise Entwicklungen im Laufe einer Zeitmessung in einem Kurvenverlauf deutlich besser zu erkennen sind als in einer CSV-Datei. Anhand der reinen Zahlen aus diesen Tabellen fallen diese wiederkehrende Trends unter anderem nicht direkt auf. Die Kennzahlen, die mithilfe von Sysbench ermittelt werden, werden in einer CSV-Datei gespeichert. Um diese tabellarische Form in eine Grafische umzuwandeln, gibt es unterschiedliche Tools, die widerum eigene Vor- und Nachteile bieten.

Die erste mögliche Alternative stellt das Tool Gnuplot Williams et al., 2024 dar. Mit diesem lassen sich CSV-Dateien sehr gut darstellen. Wenn man aber beispielweise nur bestimmte Spalten aus der Tabelle anzeigen lassen will, dann kommt man schnell an seine Granzen. Um besser Anpassungsfähig sein zu können, habe ich mich letztlich dazu entschieden ein eigenes Python-Script zu schreiben, die mithilfe der Libraries pandas (//TODO: find source) matplotlib.pyplot (//TODO: find source) die Graphen erstellt.

1.3.2 Einführung in die Tools

Als allererstes muss der MySQL-Server (oder eine anderes relationales Datenbankverwaltungssystem, das von Sysbench unterstützt wird) lokal auf dem Rechner gestartet sein. Wichtig ist dabei die User -und Passwortdaten zu merken, da diese von den Sysbench - Benchmarks benötigt werden. Nachdem das RDBMS gestartet worden ist, muss zudem eine Datenbank erstellt werden. Dies könnte unter anderem so aussehen:

```
1 CREATE DATABASE sbtest;
```

Nachdem man die Datenbank erstellt hat, muss das Tool Sysbench zunächst installiert werden. Als nächstes machen wir uns mit dem Tool und den verschiedenen Argumenten, die beim Aufruf mitübergeben werden müssen oder können, vertraut. Hier ist eine Auflistung mit den übergebenen Argumenten:

- `--db-driver`: Gibt den Treiber für die Datenbank an, die Sysbench verwenden soll. In diesem Fall `mysql`, um MySQL-Datenbanken zu testen.
- `--mysql-host`: Der Hostname oder die IP-Adresse des MySQL-Servers. Standardmäßig wird `localhost` verwendet, wenn nichts angegeben wird.
- `--mysql-user`: Der Benutzername, mit dem Sysbench auf die MySQL-Datenbank zugreift.
- `--mysql-password`: Das Passwort für den MySQL-Benutzer. Falls der Benutzer kein Passwort hat oder der Zugriff über eine andere Authentifizierungsmethode erfolgt, kann dieses Argument weggelassen werden.

- `--mysql-db`: Der Name der MySQL-Datenbank, auf die zugegriffen wird. In diesem Beispiel `sbtest`.
- `--time`: Gibt die Laufzeit des Benchmarks in Sekunden an und muss immer mit angegeben werden.
- `--report-interval`: Gibt das Intervall in Sekunden an, in dem Zwischenergebnisse während des Tests ausgegeben werden. Sofern `--report-interval` nicht gesetzt wird, werden die Ergebnisse erst am Ende des Tests angezeigt.
- `--tables`: Die Anzahl der Tabellen, die für den Test erstellt werden sollen. Standardmäßig wird nur eine Tabelle erstellt.
- `--table-size`: Die Anzahl der Datensätze (Zeilen) pro Tabelle. Muss auch nicht zwingend angegebend werden.

Neben den sieben aufgelisteten Argumenten gibt es zwei weitere wichtige Optionen:

1. Wie im Abschnitt **??** erwähnt, kann ein Lua-Skript angegeben werden, um eigene Tabellen zu erstellen, Beispieldaten einzufügen und bestimmte Abfragen durchzuführen. Dazu muss am Ende der Sysbench-Befehlszeile lediglich der Pfad zur Lua-Datei hinzugefügt werden. Ein erklärendes Beispiel dazu folgt weiter unten in diesem Abschnitt.
2. Die Methode, den Sysbench ausführen soll, muss ebenfalls spezifiziert werden. Auch dieser wird am Ende der Sysbench-Befehlszeile angehängt.

Zunächst schauen wir ein kurzes Demo-Beispiel, denn es gibt die Möglichkeit die Datenbank auf Performance zu testen, ohne selbst eigene SQL-Befehle zu schreiben. Dafür gibt es vordefinierte Testtypen von Sysbench. Auf diese Weise kann man schnell die Korrektheit der Einrichtung des Tools überprüfen, bevor man Lua-Scripts für die eigenen Bedürfnisse schreibt.

Man kann unter anderen zwischen diesen Testtypen wählen:

- **oltp_insert**: Prüft die Fähigkeit der Datenbank, Daten schnell und effizient einzufügen und simuliert eine Umgebung, in der viele Schreiboperationen ausgeführt werden.
- **oltp_read_only**: Fokussiert sich auf die Performance bei Leseoperationen und eignet sich, um die Leistung bei einer rein lesenden Arbeitslast zu testen.
- **oltp_read_write**: Simuliert eine realistische Arbeitslast, bei der sowohl Lese- als auch Schreiboperationen gleichzeitig durchgeführt werden.

Des Weiteren gibt es auch unterschiedliche Methoden, die mit den Testtypen kombiniert werden können.

-

- **prepare**: Bereitet die Datenbank für den Test vor, u.a. das Einfügen von benötigten Datensätze.
- **run**: Ist die Ausführungsphase des Tests. Je nach Testtyp führt diese Methode die spezifizierten Operationen aus, wie etwa das Einfügen von Daten (oltp_insert), das Abfragen von Daten (oltp_read_only) oder beides (oltp_read_only). Dabei wird die Performance der Datenbank unter der angegebenen Arbeitslast gemessen.
- **cleanup**: Diese Methode sorgt dafür, dass nach Abschluss des Tests alle Testdaten entfernt werden. Sie stellt die Datenbank in ihren ursprünglichen Zustand zurück und stellt sicher, dass keine Testdaten zurückbleiben, die eine mögliche produktive Umgebung beeinträchtigen könnten.

Für das Demo-Beispiel wählen wir den Testtypen **oltp_read_write** und allen Methoden aus. Für die Methode run würde unsere Query so aussehen, wobei `YOUR_USER` und `YOUR_PASSWORD` entsprechend ersetzt werden müssten:

```
sysbench oltp_read_write \
  --db-driver=mysql \
  --mysql-user=YOUR_USER \
  --mysql-password=YOUR_PASSWORD \
  --mysql-db="sbtest" \
  --time=10 \
  --report-interval=1 \
  run
```

Wenn man nur diese Query ausführt, fällt er auf, dass die Query scheitert. Deshalb bietet es sich an ein Shell-Script zu schreiben, indem zuerst `prepare` aufgerufen wird und als Nächstes erst `run`. Die Ergebnisse der Log-Datei speichert man sich dann in einer Datei und aus dieser Datei erstellt man eine CSV-Datei, mit der man später die Graphen erstellen lässt. Und als letzten Schritt ruft man die `cleanup`-Methode auf, damit bei erneuter Ausführung keine Fehler entstehen, bzw. die Produktivumgebung nicht gestört wird, wenn diese sonst beeinflusst werden würde.

Dies ist das Shell-Script, dass zuständig ist für den kompletten Ablauf:

Codeblock 1.1: Sysbench Script

```
#!/bin/bash

# File Paths
GENERATE_PLOT_SCRIPT="/Users/danielmendes/Desktop/Bachelorarbeit/Ausarbeitung/Tools/
    Pandas/generateplot.py"
```

```
OUTPUT_DIR="output"
OUTPUT_FILE="output/sysbench_output.csv"
RAW_RESULTS_FILE="output/sysbench.log"
GNUPLOT_SCRIPT="plot_sysbench.gp"

# Connection parameters
DB_USER="root"
DB_PASS="password"
DB_NAME="sbtest"
TABLES=10
TABLE_SIZE=10000
DURATION=10

# Ensure output directories exist
rm -rf "$OUTPUT_DIR"
mkdir -p "$OUTPUT_DIR"

# Function to run sysbench with parameters
run_sysbench() {
  local MODE="$1"
  local EXTRA_ARGS="$2"
  local LOG_FILE="$3"

  sysbench oltp_read_write \
    --db-driver=mysql \
    --mysql-user="$DB_USER" \
    --mysql-password="$DB_PASS" \
    --mysql-db="$DB_NAME" \
    --tables="$TABLES" \
    --table-size="$TABLE_SIZE" \
    $EXTRA_ARGS \
    $MODE >> "$LOG_FILE" 2>&1

  return $?
}

echo "Preparing the database..."
run_sysbench "prepare" "" "$RAW_RESULTS_FILE"
echo "Database prepared."

echo "Running benchmark..."
run_sysbench "run" "--time=$DURATION --threads=1 --report-interval=1" "
```

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

# Format the results into CSV
echo "Time (s),Threads,TPS,QPS,Reads,Writes,Other,Latency (ms;95%),ErrPs,ReconnPs" > "$OUTPUT_FILE"
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 "Cleaning up..."
run_sysbench "cleanup" "" "$RAW_RESULTS_FILE"
echo "Database cleanup complete."

# Generate plot with gnuplot
rm -rf "$OUTPUT_DIR/gnuplot"
mkdir -p "$OUTPUT_DIR/gnuplot"
echo "Generating plot with gnuplot..."
gnuplot $GNUPLOT_SCRIPT

# Generate plot with pandas and move objects
echo "Generating plots with pandas..."
python3 "$GENERATE_PLOT_SCRIPT" "$OUTPUT_FILE"
SOURCE_DIR="output/detailed_pngs"
DEST_DIR="output/pandas"
FILE_TO_MOVE="output/output_final.png"
```

```
mkdir -p "$DEST_DIR"
mv "$SOURCE_DIR"/* "$DEST_DIR"
mv "$FILE_TO_MOVE" "$DEST_DIR/Summary.png"
rm -rf "$SOURCE_DIR"

echo "Plots generated."
```

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
```

Das Python-Script, das zuständig ist für die Graphgenerierung muss als Argument zum einen die CSV-Datei übermittelt bekommen und zum anderen kann es nur eine bestimmte Auswahl an Messwerten übergeben, damit nur für diese die Graphen erzeugt werden. Dies ist das zuständige Python-Script:

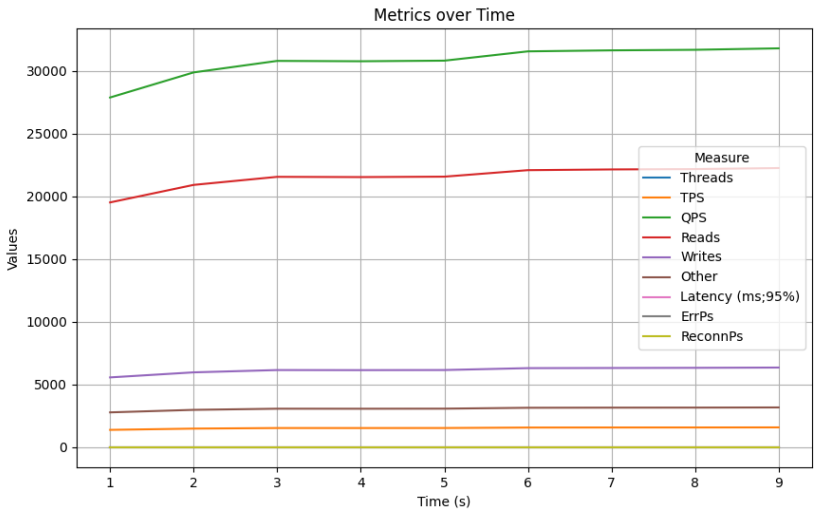
Codeblock 1.3: Pandas Graph Generator

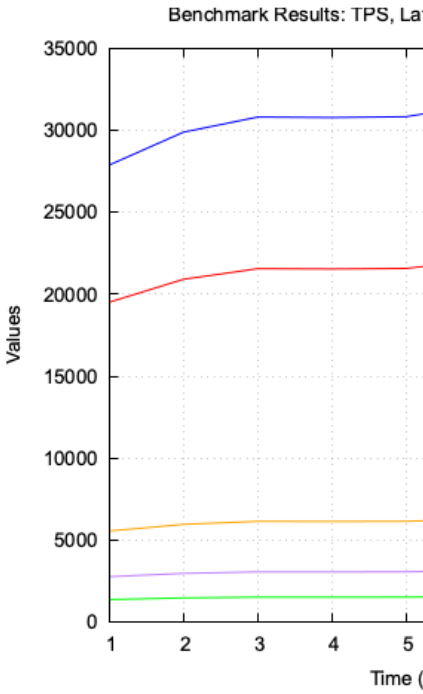
```
import pandas as pd
import matplotlib.pyplot as plt
import os
```

```
import argparse

def parse_arguments():
    parser = argparse.ArgumentParser(description='Generate plots from CSV data.')
    parser.add_argument('datafile', type=str, help='Path to the input CSV data file')
    parser.add_argument('metrics', type=str, nargs='*', help='List of metrics to plot (e.g., QPS Reads Writes). If empty, all metrics will be used.')
    return parser.parse_args()

def plot_metrics(data, measures, output_dir, detailed_pngs_dir):
    has_script_column = 'Script' in data.columns

    if has_script_column:
        scripts = data['Script'].unique()
    else:
        scripts = [None]

    plt.figure(figsize=(10, 6))

    for measure in measures:
        if has_script_column:
            # Plot each script as a separate line for each measure if 'Script' column exists
            for script in scripts:
                script_data = data[data['Script'] == script]
                plt.plot(script_data['Time (s)'], script_data[measure], label=f"{script} - {measure}")
        else:
            # Plot only the measure if no 'Script' column exists
            plt.plot(data['Time (s)'], data[measure], label=measure)

    plt.title('Metrics over Time' + (' by Script' if has_script_column else ''))
    plt.xlabel('Time (s)')
    plt.ylabel('Values')
    plt.legend(title="Script and Measure" if has_script_column else "Measure")
    plt.grid(True)

    # Save the combined plot
    output_final_path = os.path.join(output_dir, 'output_final.png')
    plt.savefig(output_final_path)
    plt.close()
```

```
42
43      for measure in measures:
44          plt.figure(figsize=(10, 6))
45          if has_script_column:
46              # Plot each script as a separate line for each measure if 'Script'
    column exists
47              for script in scripts:
48                  script_data = data[data['Script'] == script]
49                  plt.plot(script_data['Time (s)'], script_data[measure], label=f"{
    script} - {measure}")
50          else:
51              # Plot only the measure if no 'Script' column exists
52              plt.plot(data['Time (s)'], data[measure], label=measure)
53
54          # Plot settings for individual figures
55          plt.title(f'{measure} over Time by Script')
56          plt.xlabel('Time (s)')
57          plt.ylabel(measure)
58          plt.legend(title="Script")
59          plt.grid(True)
60
61          # Save the detailed plot to a PNG file in the specified output directory
62          detailed_output_file_path = os.path.join(detailed_pngs_dir, f"{measure}.png"
    )
63          plt.savefig(detailed_output_file_path)
64          plt.close()
65
66  def main():
67      args = parse_arguments()
68
69      # Load CSV data
70      datafile = args.datafile
71      if not os.path.isfile(datafile):
72          raise FileNotFoundError(f"The file {datafile} does not exist.")
73
74      data = pd.read_csv(datafile)
75
76      # Determine metrics to plot
77      if args.metrics:
78          measures = args.metrics
79      else:
80          # Use all columns except 'Time (s)' and 'Script' as metrics
```

```
81          measures = [col for col in data.columns if col not in ['Time (s)', 'Script'
    ]]
82
83      # Define output directory for plots
84      output_dir = os.path.dirname(datafile)
85      detailed_pngs_dir = os.path.join(output_dir, 'detailed_pngs')
86
87      # Create output directories if they don't exist
88      os.makedirs(detailed_pngs_dir, exist_ok=True)
89      plot_metrics(data, measures, output_dir, detailed_pngs_dir)
90
91  if __name__ == '__main__':
92      main()
```





- **Threads**: Die Anzahl der gleichzeitig verwendeten Threads. Mehr Threads können die Parallelität erhöhen, jedoch kann eine zu hohe Anzahl die Leistung beeinträchtigen, wenn das System überlastet wird.
- **TPS (Transactions Per Second)**: Die Anzahl der Transaktionen pro Sekunde. Ein höherer Wert deutet auf eine bessere Datenbankleistung hin.
- **QPS (Queries Per Second)**: Die Anzahl der Abfragen pro Sekunde. Ein höherer Wert ist besser und zeigt die Effizienz bei der Verarbeitung von Abfragen.

- **Reads**: Die Anzahl der Leseoperationen. Mehr Leseoperationen sind im Allgemeinen besser, da sie eine höhere Datenauslastung anzeigen, was jedoch auch vom spezifischen Anwendungsfall abhängt.
- **Writes**: Die Anzahl der Schreiboperationen. Ähnlich wie bei den Leseoperationen: Mehr Schreibvorgänge sind besser, solange die Performance erhalten bleibt.
- **Other**: Bezieht sich auf andere Arten von Operationen, die weder als Reads noch als Writes kategorisiert werden. Ein höherer Wert ist gut, solange er nicht zu einer Überlastung führt.
- **Latency (ms; 95%)**: Die durchschnittliche Zeit in Millisekunden, die benötigt wird, um Anfragen zu bearbeiten, wobei der Wert im 95. Perzentil betrachtet wird. Niedrigere Werte sind besser, da sie auf schnellere Reaktionszeiten hinweisen.
- **ErrPs (Errors Per Second)**: Die Anzahl der Fehler pro Sekunde. Ein niedriger Wert ist wünschenswert, da er auf eine höhere Stabilität und Zuverlässigkeit des Systems hinweist.
- **ReconnPs (Reconnects Per Second)**: Die Anzahl der Wiederverbindungen pro Sekunde. Ein niedrigerer Wert ist ebenfalls besser, da häufige Wiederverbindungen auf Stabilitätsprobleme hindeuten können.