Deutung der Ergebnisse

- einige Ausreißer in den Messreihen auf Laptop und PC,
  verursacht durch kurzzeitige Lastspitzen bei Betriebssystem und/oder Hardware
- großteil der Messwerte auf einem Niveau,
  wahrscheinlich Caching der Werte, dadurch schnellere Zugriffe und Operationen
- Messwerte beim Laptop sprunghaft und unregelmäßig,
  wahrscheinlich wegen anderer CPU Architektur oder Kühlung bzw. Stromzufuhr
- auch bei den 1. Messwerten ist keine Datenstruktur klar vorne,
  somit unklar welche Datenstruktur schneller ist
- wahrscheinlich andere Limitation der Geschwindigkeit (z. B. RAM Zugriff)
- unterschiedliche Messwerte bei Daten gleicher Datenstruktur

Beantwortung der Forschungsfrage

Abschließend lässt sich die Forschungsfrage insofern beantworten, als dass die Wahl der Datenstruktur in diesem Fall keinen Einfluss auf die Zugriffsgeschwindigkeit hat. Keine der Datenstrukturen scheint der anderen in Geschwindigkeit überlegen, zumindest auf Basis der Messwerte. Um wirklich sicher zu gehen und noch aufschlussreichere Ergebnisse zu erhalten wären aber noch weitere Test von Vorteil.

Schwierigkeiten und Mögliche Verbesserungen

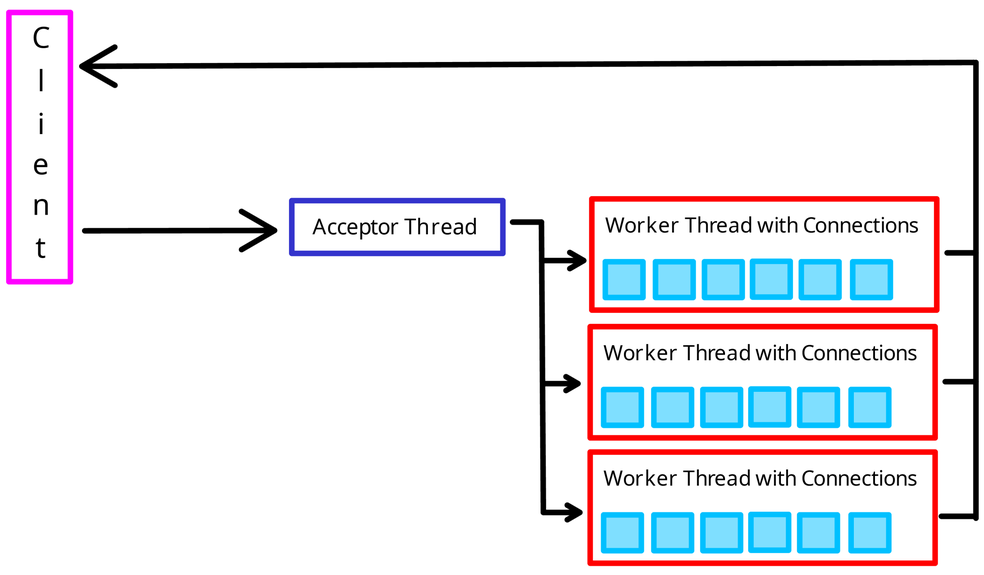
- Umsetzung von Datenbank und Datenstrukturen teil recht schwer,
  evtl. woanders abschauen
- genauere Performance-Analyse mit CPU Performance Countern,
  AMDµProf lief leider nicht auf meinem PC
- Testen mit SSD, HDD und RAM für mehr Aufschlüsse

Github

DB Engines Overview

BJ Network Guide

Aktueller Stand meiner “Datenbank”




```
00000000  01 00 01 00 04                                    ·◇·◇·
```

```
00000000  01 00 01 00 04 04 00 00  00 04 00 00 00 ff ff ff  ·◇·◇··◇◇  ◇·◇◇◇×××
00000010  ff ff ff ff ff 00 00 00  00 00 00 00 00 04 00 00  ×××××◇◇◇  ◇◇◇◇◇·◇◇
00000020  00 17 00 00 00 00 00 00  00 00 00 00 00 05 00 00  ◇·◇◇◇◇◇◇  ◇◇◇◇◇·◇◇
00000030  00 17 00 00 00 00 00 00  00 00 00 00 00 06 00 00  ◇·◇◇◇◇◇◇  ◇◇◇◇◇·◇◇
00000040  00 17 00 00 00 00 00 00  00 00 00 00 00 07 00 00  ◇·◇◇◇◇◇◇  ◇◇◇◇◇·◇◇
00000050  00 17 00 00 00 00 00 00  00 00 00 00 00           ◇·◇◇◇◇◇◇  ◇◇◇◇◇
```

Struktur der Datenbank

- Anfragen werden auf die Worker-Threads aufgeteilt, optimale CPU Auslastung
- Nutzung von Queues und Locks für Datenintegrität
- effiziente Eventarchitektur mit **poll** und **epoll**

Struktur des binären Speicherformats

- Binärdatei in Hexadecimaldarstellung
- Header (oben) stellt Struktur dar, legt Datentypen und größen Fest
- Datei mit Daten befüllt (unten)
  Schlüssel jeweils 4,5,6,7 und als Daten 0x17
- Zwischenräume für Pointer/Offsets

Bug im C++ Client

- Python Client geht
- Code funktioniert nur mit Zeilen 27/28 auskommentiert
- Failbit von **std::cin** wird aus unbekanntem Grund gesetzt
- **std::getline** blockiert nicht
- hoffe auf **Bugfix + Erklärung**

```cpp
int main(int argc, char *argv[]) {
    Client client("127.0.0.1", PORT);

    std::string query_string = "";
    std::cout << "lightdb>";
    std::cin.exceptions(std::ios_base::failbit);
    while (true) {
        try {
            std::getline(std::cin, query_string, '\n');
        } catch (std::ios_base::failure &e) {
            std::cout << e.what() << "\n";
        }

        if (check_query_validity(query_string)) {
            // client.Query(query_string);
            // client.Result();
        } else {
            std::cout << "Error in Query String!\n";
        }
        std::cout << "lightdb>";
    }

    return 0;
}
```

Hier ist noch Platz, also eine Meme

Anton Rodenwald, Schillerschule Hannover

Schillerschule Hannover