Verteilte Systeme, Übungsblatt 1, Sommer 2024

Abgabe: Benedikt Maria Beckermann, Mtr.Nr. 1599203, 27.05.2024

- Link zum Repository: https://github.com/greenBene/24S-DS-A01

Aufgabe 1

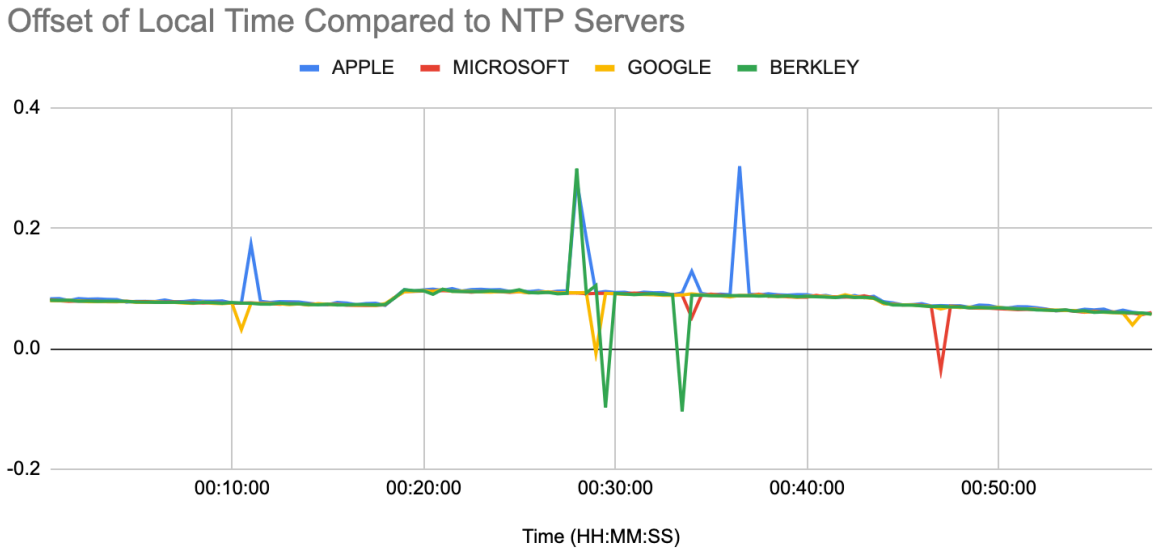

Abbildung 1: Abweichung Zwischen der Lokale Uhr und NTP Servern von Apple, Microsoft, Google, und Berkley, über eine Stunde

In Aufgabe 1 sollten wir uns überlegen, wie wir über einen Zeitraum von mindestens einer Stunde die Zeit auf unserem lokalen Rechner mit der Zeit auf einem oder mehreren Zeitservern vergleichen und diesen Ansatz auch umsetzen.

Ich habe mich entschieden diese Aufgabenstellung über ein Python Script zu lösen, welches die Bibliothek ntplib[1] benutzt, um mit verschiedenen NTP-Servern zu kommunizieren. Das Script speichert die Ergebnisse in einer CSV Datei. Das Script stellt alle 30 Sekunden Anfragen an die NTP-Server von Apple, Microsoft, Google, und der UC Berkeley. Basierend auf den Antworten berechnet die ntplib Bibliothek automatisch den Offset Value nach der Formel, die auch in der Vorlesung besprochen wurde. Da vor allem dieser Offset, der die Ungenauigkeit der lokale Uhr zeigt, für diese Aufgabe relevant ist, wird auch nur dieser Wert jeweils in die Ergebnis CSV ausgegeben. Die Ergebnisse eines Runs sind sowohl als CSV Datei im Repository unter den Pfad "Aufgabe01/aufgabe01_results.csv" zu finden, als auch grafisch dargestellt in der obenstehenden Abbildung. Das benutzte Script ist im Repository unter "Aufgabe01/aufgabe1.py" zu finden.

[1] https://pypi.org/project/ntplib/, accessed 2024-05-26

Es fällt auf, dass, bis auf die Ausnahme von ein paar Outliers, die Offsets zwischen den verschiedenen Zeitservern relativ ähnlich sind. Dies ist zu erwarten, da die Zeitserver selbst eine deutlich höhere Zeitgenauigkeit haben als mein Rechner, wodurch die Zeitabweichung zwischen meinem Rechner und den Zeitservern deutlich höher ist, als die Zeitabweichung zwischen den verschiedenen Zeitservern.
Weiter fällt auf, dass zu den Zeitpunkten 00:19:00 und 00:43:00 sich der Offset zu allen vier Zeit-Servern verändert und auf dem neuen Wert relativ konstant bleibt. Ich gehe davon aus, dass zu diesen Zeitpunkten mein Rechner die lokale Uhr mit einem Zeitserver synchronisiert hat.
Die erwähnten Outliers kann ich nicht basierend auf der Funktionsweise des Network Time Protokoll erklären, aber ich gehe davon aus, dass es hier zu Schwankungen durch die Python Environment kam und ein ähnliches Experiment, welches in einer systemnahen Sprache implementiert wurde weniger dieser Outliers aufweist.

Aufgabe 2

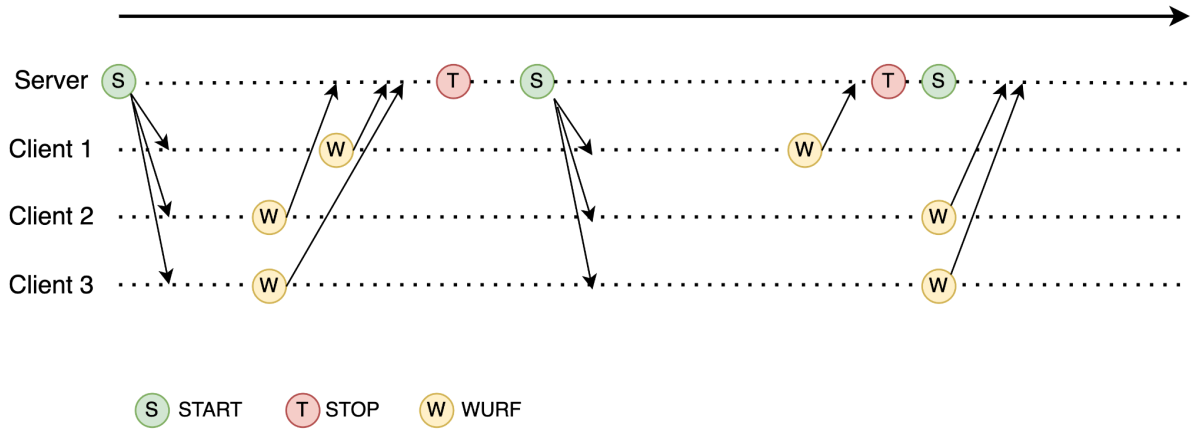

Abbildung 2: Schematische Darstellung der Ausgabe des Systems von Aufgabe 2a

Aufgabe 2a

In Aufgabe 2a sollten wir ein Würfelspiel implementieren, bei dem es einen Server und mehrere Klienten gibt, die alle über ein Netzwerk miteinander kommunizieren, aber keine Zeitstempel oder Ähnliches in ihren Nachrichten mit senden dürfen.

Die Lösung hierfür habe ich über zwei Python-Skripte implementiert, die über Sockets mithilfe des TCP Protokolls kommunizieren. Um zu simulieren, dass die Scripte auf verschiedenen Rechnern laufen, habe ich das ganze über verschiedene Docker Container laufen gelassen. Den Python Code, die Docker Konfiguration, und auch eine Beispiel-Ausgabe von Docker Compose kann im Repository im Pfad "Aufgabe02a/" gefunden werden.

Wie zu erwarten, kommt es bei der Ausführung zu Synchronisierungsproblemen. Die Klienten verschicken ihre WURF Ergebnisse teilweise erst nachdem der Server bereits die Gewinner*in gewählt und die nächste Runde gestartet hat. Da der Server allerdings nicht unterscheiden kann, zu welcher Runde eine WURF Nachricht gehört, kann es passieren, dass ein WURF Ergebnis, welches erst nach dem Start der nächsten Runde angekommen ist, für die neue Runde gewertet wird, zu der es allerdings nicht gehört. Dies ist auch schematisch in Abbildung 1 zu sehen.

Durch das Requirement, dass das System auf verschiedenen Rechner laufen soll, nimmt der Aufwand der Implementierung und vor allem der Orchestrierung deutlich zu. Es muss darauf geachtet werden, dass die jeweiligen Rechner im selben Netzwerk sind und auch alle benötigten Ports offen sind.

Aufgabe 2b

In Aufgabe 2b soll das selbe Problem wie in Aufgabe 2a gelöst werden, allerdings mit Hilfe von logischen Uhren.

Hierzu habe ich eine Vektor Uhr Implementiert, und sowohl den Klienten als auch den Server aus Aufgabe 2a erweitert, dass sie diese Vektor Uhr lokal instanziieren, bei lokalen Ereignissen ihre Zeit erhöhen, ihre Vektor Zeiten in den Nachrichten mit senden, und beim Empfangen einer Nachricht ihre eigene Zeit erhöhen und abschließend mit der empfangenen Zeit entsprechend updaten. Der Server testet außerdem beim Empfang einer WURF Nachricht, ob diese kausal abhängig vom letzten START Ereignis ist, und akzeptiert den Wurf nur, wenn dies der Fall ist. Der Code samt Beispiel-Ausgabe ist im Repository unter dem Pfad "Aufgabe02b/" zu finden.

In diesem Experiment werden WURF Nachrichten, die zu spät beim Server ankommen, verworfen, anstatt, dass sie der falschen Runde zugewiesen werden. Interessant hier ist auch, dass die Zeitstempel der Logging Informationen teilweise leicht von der Reihenfolge der Nachrichten basierend auf den Vektor-Uhren abweicht. Dies ist durch das Blockieren einzelner Threads zu erklären. Durch die Vektor-Uhren ist jedoch sichergestellt, dass unabhängig vom temporären Blockieren der Threads es zu keinen invaliden Zuständen kommen kann, in denen ein Wurf einer falschen Runde zugewiesen wird.

Der Aufwand, das System auszuführen, war vergleichbar mit dem Aufwand von Aufgabe 2a.

Aufgabe 2c

In Aufgabe 2c soll das System aus Aufgabe 2b lokal mit möglichst vielen Klient-Instanzen ausgeführt werden. Das Programm wurde hier sehr langsam und die meisten WURF Nachrichten kamen erst nach dem Ende der jeweiligen Runden an. Durch die Vektor-Uhren wurden diese Nachrichten vom Server verworfen.

Die Performance ist vermutlich durch eine nicht optimale Implementierung zu erklären. Der Aufwand, die einzelnen Klienten manuell und lokal zu erstellen, hielt sich in Grenzen, da

lediglich ein neues Terminal geöffnet werden musste. Wenn aber manuell so viele Klienten auf unterschiedlichen Rechnern gestartet werden müssten, würde der Aufwand hierzu drastisch zunehmen, solange keine Automatismen, wie Docker Compose, benutzt werden.

Um das System unter Last zu testen, wurde es zusätzlich auch mit 50 Instanzen via Docker Compose gestartet. Die Logs davon sind unter "Aufgabe02b/results02c.txt" zu finden. Ähnlich wie bei der lokalen Ausführung nahm die Performance ab. Durch das automatische Starten der 50 Container durch Docker Compose war hier aber der Aufwand deutlich kleiner als beim manuellen Starten.