Bonusaufgabe: Erweiterung

L. Conte

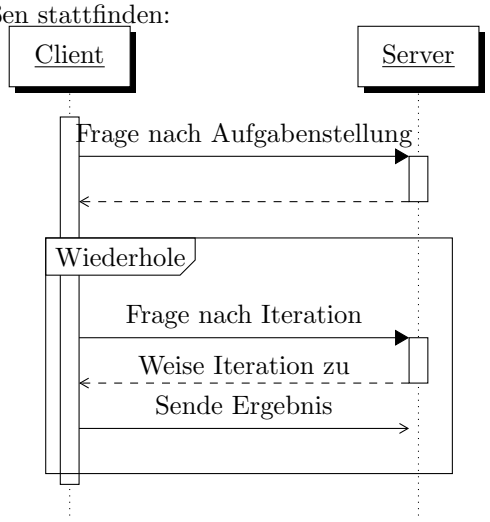
April 2022

1 Einführung

Die verschiedenen Iterationen des Lösungsansatzes werden bereits in der Standardlösung parallel verarbeitet. Dabei wird jedoch immer nur ein Computer gleichzeitig verwendet. Die folgende Erweiterung ist eine Möglichkeit, den Rechenaufwand auf mehrere in einem Netzwerk liegende Computer zu verteilen.

2 Idee

Die Teilnehmer an der verteilten Berechnung sind ein Server und mindestens ein Client. Die Clients führen die eigentliche Suche aus, während der Server die Aufgabenstellung an die Clients verteilen soll und den Clients Iterationen zur Berechnung zuweisen soll.
Für dieses einfache Modell, in dem die Clients nichts voneinander wissen müssen reicht, ein Protokoll, wie HTTP aus. Die Kommunikation kann dann folgendermaßen stattfinden:



Ein Client kann mehrere Iterationen parallel verarbeiten und ein Computer kann als Client und Server gleichzeitig funktionieren.

Die Daten, wie zum Beispiel die Aufgabenstellung, werden als JSON Text übertragen. Es gibt selbstverständlich effizientere Formate, doch die meiste Zeit wird bei der Verarbeitung und nicht der Kommunikation verbracht.

Was jetzt noch fehlt ist eine Strategie, nach der die Iterationen zugewiesen werden. Am Anfang sollen Iterationen aufsteigend verteilt werden. Anstatt keine weiteren zu verteilen, nachdem man am Ende angekommen ist, werden bereits zugewiesene aber bisher unbeantwortete Iterationen neu an neue Anfragen zugewiesen. Dadurch wird verhindert, das die Verarbeitung hängen bleibt, nachdem ein Client crasht oder zu lange für die Verarbeitung braucht.

3 Umsetzung

Die Umsetzung ist in Rust geschrieben und auf zwei Dateien "client.rs" und "server.rs" verteilt. Zusätzlich zu den Verarbeitungsfunktionen der Standardlösung werden die Pakete hyper, serde und serde_json verwendet. Die zwei serde Pakete bieten eine einfache Möglichkeit, bereits existierende Strukturen, wie *TInput*, als JSON zu senden. hyper bietet alles notwendige für die Verwendung von HTTP und ist wiederum von tokio, einer async Laufzeitumgebung, abhängig.
Der Server auf Port 3000 akzeptiert folgende Routen:

- GET /tinput für die Aufgabenstellung
- GET /get_assignment für eine Zuweisung
- POST /assignment_result für das Ergebnis einer Iteration

4 Beispiele

Es folgen die Laufzeiten für Beispiele von bwinf.de. Zur Berechnung wurden verwendet:

- Windows Laptop mit 4 Threads
- Windows Laptop mit 4 Threads
- Macbook Air mit 1 Thread

Die weitere Ausgabe ist hier nicht enthalten, da sie selbstverständlich die gleiche, wie für das Standardprogramm ist.

- Beispiel 2: ca. 8s
- Beispiel 3: ca. 83s
- Beispiel 4: ca. 563s

Argumente:
Client: `<exe> <Server url> <Threadzahl> <Speichergröße/10^7>`
Server: `<exe> <Eingabedatei>`

5 Probleme und Erweiterungen

Das vorgestellte System ist eine einfache Lösung für die verteilte Lösung einer Aufgabenstellung und könnte an verschiedenen Punkten noch erweitert werden.

- Bisher kann nur eine einzige Aufgabenstellung gelöst werden und der Server muss vor den Clients gestartet werden. Mit anderen Protokollen als HTTP könnte eine "dynamischere" Kommunikation ermöglicht werden, bei der der Server den Client benachrichtigt, wenn neue Iterationen berechnet werden müssen.
- Die Kommunikation könnte noch gesichert werden, so dass sich nicht jeder beliebige Client mit einem Server verbinden kann.