Aufgabe 3

Emanuela Giovanna Calabi 13186
Davide Pizzirani 13413
Ravinder Singh 13642

Verteilter MD5-Cracker: Dokumentation Teil 2

Einleitung

Unser Hauptziel bei der Implementierung des MD5-Crackers war es, die Robustheit in den Vordergrund zu stellen, während die Geschwindigkeit an zweiter Position stand. So konnten wir ein System erreichen, das völlig offen und ausfallsicher ist. Wir haben das Feedback zum ersten Teil berücksichtigt. Insbesondere erkannten wir die Schwierigkeit, alle Clients auf dem nextNumberToCrack zu synchronisieren, oder auf ein neues Problem, wenn es vom Server gesendet wurde oder das aktuelle vorbei war. Oftmals kam es vor, dass die Kunden die gleiche Nummer berechneten, oder dass sie sich stoppten, wenn ein neues Problem auftrat. Aus diesen Gründen und dank der Fristverlängerung haben wir uns dafür entschieden, ganz neu anzufangen und einen anderen Lösungsansatz zu wählen, der es auch ermöglicht, den Kommunikationsaufwand zu reduzieren.

1. Kommunikation zwischen den Clients

Der wichtigste Punkt unseres Projekts ist, wie die Kunden miteinander kommunizieren. Um Nachrichten zwischen allen Clients auszutauschen, haben wir ein verteiltes Log implementiert, das alle Informationen über alle Ereignisse enthält, die zwischen den Clients und zwischen den Clients und dem Server in Echtzeit stattfinden. Jeder Log-Eintrag (LogEntry) kann bedeuten, dass ein neuer Client unserem Netzwerk beigetreten ist, oder dass ein Problem veröffentlicht wurde, oder dass Client X den Wertebereich Y geknackt hat, usw. . Zu diesem Zweck haben wir das Objekt LogEntry implementiert und unser verteiltes Log ist einfach ein ArrayList<LogEntries>. Zum Beispiel könnte es so etwas wie das folgende aussehen:

[94] rmi://localhost/server/ PROBLEM 10000000 a05bb1cd219a84a2a...
...
[98] 10.7.145.175:2002/client2 START 50000
…
[101] 10.7.145.175:2002/client2 DONE 50000

Jeder Kunde wird auf dem Laufenden gehalten über alle Aktivitäten der anderen Kunden. Auf diese Weise, wenn ein Client, der an einer Teilmenge des Problems arbeitet, unterbrochen wird, bevor es fertig ist, können die anderen aktiven Clients die Arbeit des toten Clients wiederherstellen. Die Nachrichtenübermittlung erfolgt über die Übergabe eines LogEntry-Objekts an die anderen Clients über RMI: Ein Client ruft die Methode broadcast() auf, die intern einen RMI-Aufruf verwendet, um auf jedem der anderen Clients writeLog() zu schreiben. Abhängig vom Inhalt dieser Nachricht werden die Clients dann aktiv. Zum Beispiel, wenn ein neuer Client dem Netzwerk beigetreten ist, werden die anderen Clients ihn zu HashMap hinzufügen, der die Namen und Adressen der Clients enthält, oder wenn ein Client eine Lösung für das aktuelle Problem gefunden hat, sollten alle anderen Clients anhalten und auf einen neuen warten. Dies mag zwar viel Aufwand für die Kommunikation bedeuten, aber es erlaubt uns, den verschiedenen Clients größere Teilprobleme zuzuordnen, was weniger Kommunikation bedeutet, und trotzdem sicherzustellen, dass jede Zahl berechnet wird.

2. Strategie zur Problemteilung

Wenn ein Problem vom Server veröffentlicht wird und der Master-Client es empfängt, wird eine neue LogEntry-Nachricht für das Problem erstellt und an alle anderen Clients gesendet. Aus Synchronisationsgründen startet der Master-Client zuerst und sendet dann eine Nachricht, dass er mit dem Crack gestartet ist, damit auch andere Clients starten können. Jeder Client arbeitet jeweils an einer Teilmenge des Problems, beginnend bei 0 bis zur vom Server gesendeten Problemgröße. Jede Teilmenge ist ein Teilbereich von 0 bis zur Problemgröße und ihre Größe hängt von dem innerhalb der Clients definierten Variablen range ab. Das Subproblem wird dann vom WorkerThread berechnet, um den Client für die Kommunikation zur Verfügung zu stellen. Wenn ein Client mit dem Knacken des aktuellen Bereichs fertig ist, sendet er ihn an die anderen Clients, damit diese wissen, dass dieser Bereich getestet wurde. Jeder Client wählt selbstständig einen zu knackenden Bereich basierend auf dem Inhalt des Log. Wenn eine Lösung gefunden wird, wird der WorkerThread gewartet und der Client teilt sie den anderen Clients mit, damit sie alle aufhören können. Der Client, der die Lösung gefunden hat, meldet sich dann erneut beim Server an und sendet sie. Wir registrieren uns jedes Mal neu, wenn wir eine Lösung einreichen, da unser System Peer-to-Peer-basiert ist und daher keinen Master per se benötigt.

3. Mechanismen zur Offenheit und Ausfallsicherheit

Unser Ansatz, ein verteiltes Log zu verwenden, hat es uns ermöglicht, Offenheit und Ausfallsicherheit zu erreichen. Jeder Klient kann jederzeit beitreten oder gehen: Solange er noch aktiv ist, steht unser System und bietet einen Verbindungspunkt für potentielle Klienten. Wenn ein Client beitritt, während die anderen bereits arbeiten, erhält der neue Client die HashMap mit den Daten der anderen Clients und der LogEntry mit dem neuesten Problem, falls vorhanden. Dann sendet der neue Client, dass er beigetreten ist, so dass die

alten Clients ihre HashMaps mit allen Clients im Netzwerk aktualisieren können. Der neue Client wählt dann einen Bereich aus und beginnt zu arbeiten. Wenn alle Bereiche getestet wurden, aber noch keine Lösung gefunden wurde, suchen die Clients im Protokoll nach allen Bereichen, die nicht als erledigt markiert wurden. Dies sagt uns, dass ein Client an diesem Punkt untergegangen ist und seine Aufgabe nicht erfüllt werden konnte. Dies stellt sicher, dass alle möglichen Nummern für eine Lösung getestet werden und wenn ein Client unerwartet ausfällt, ist das kein Problem.