SUBSCALE Outline

William Mendat[1] Max Ernst[2] Steven Schall[3] Matthias Reichenbach[4]

1 Verteilung

1.1 CoreSets

Der Algorithmus beginnt damit, die CoreSets jeder Dimension zu bilden. Die Bildung der CoreSets geschieht innerhalb einer Dimension und ist zudem auch komplett unabhängig von den anderen Dimensionen. Dabei wird innerhalb einer Dimension jeder Punkt und die Entfernungen dieses Punktes zu anderen Punkten betrachtet. Sollte die Entfernung kleiner als der angegebene Epsilon-Wert sein, so wird der Punkt zum CoreSet hinzugefügt.

Da in diesem Prozess keine Abhängigkeit zu anderen Dimension besteht und lediglich die Daten der jeweiligen Dimension benötigt werden, kann die Bildung der CoreSets für jede Dimension parallel abgearbeitet werden.

Somit kann erreicht werden, dass jeder PC in einem Cluster nur ein Subset von Dimensionen abarbeitet, um die Corsets zu bilden.

1.2 DenseUnits

Eine Dense Unit beschreibt die Menge aller Kombinationen von Punkten in einem Core Set als Tupel mit der Größe der minimalen Core Set Größe. Die Berechnung der Dense Units ist für jedes Core Set unabhängig.
Sind die Core Sets ermittelt, so können diese an unabhängige Recheneinheiten distributiert werden. Die Recheneinheit erstellt dann alle Tupel des Core Sets und persistiert diese in einem geeigneten Medium.
Die Dense Unit Bildung selbst kann ebenfalls weiter parallelisiert werden. Die Kombination der im Core Set enthaltenen Punkte erfolgt sequentiell nach Schema: Bei einer minimalen Core Set Größe von drei Punkten wird für die Bildung der Dense Units zunächst der Punkt 1 und 2 mit dem Punkt 3, dann 4, dann 5 usw. kombiniert: (1, 2, 3), (1, 2, 4), (1, 2, 5), ... Sind

[1] Hochschule Offenburg, Offenburg, Deutschland wmendat@stud-hs.offenburg.de
[2] Hochschule Offenburg, Offenburg, Deutschland mernst@stud-hs.offenburg.de
[3] Hochschule Offenburg, Offenburg, Deutschland sschall@stud-hs.offenburg.de
[4] Hochschule Offenburg, Offenburg, Deutschland mreichen@stud-hs.offenburg.de

alle Kombinationen, die die Punkte 1 und 2 enthalten ermittelt, folgen alle Kombinationen, die die Punkte 1 und 3 enthalten: (1, 3, 4), (1, 3, 5), ... Das Intervall kann auf mehrere Recheneinheiten verteilt werden. Die Recheneinheiten ermitteln dann jeweils die Dense Units aus ihrem Intervall und legen diese im gemeinsamen Speicher ab.

1.3 System vergleicht Dense Units von Zwei Dimensionen

Nachdem die Dense Units für eine Dimension und deren CoreSets erzeugt wurden, müssen für diese Dense Units Signaturen gebildet werden und die Punkte mit deren Dimension in eine Tabelle eingetragen werden. Die Berechnung der Signatur ist unabhängig von anderen Dimensionen und Dense Units, somit kann dies auf verschiedene Rechnern Verteilt werden. Das eintragen der Daten muss jedoch über Rechnergrenzen hinweg synchronisiert werden. Wenn ein Rechner für die übermittelte Dense Unit die Signatur erzeugt, muss dieser vorerst prüfen ob bereits ein Eintrag mit dieser Signatur existiert und ergänzt die Dimension an diesem Eintrag. Wenn beispielsweise für die Datenhaltung eine SQL oder NoSQL Datenbank verwendet wird, muss bevor ein Eintrag gespeichert wird, zuerst abgefragt werden ob die Signatur bereits hinterlegt ist. Bei großen Datenmengen erzeugt das ständige abfragen ob die Signatur bereits existiert einen großen overhead beim speichern. Eine mögliche Alternative um diese Abfrage zu umgehen, kann eine eigene Implementierung einer Distributed Hash Table verwendet werden. Dabei werden die Dense Units sowie die Dimension dem Peer-to-Peer Netzwerk übergeben, dieses berechnet aus den Punkten die Summe (Hashwert) der Dense Unit und beauftragt den zuständigen Knoten die Daten zu speichern. Der Zuständige Knoten müsste bei einer Kollision der Signatur die Dimension am Eintrag ergänzen. Dadurch würde das berechnen der Signatur auf dem Peer-to-Peer Netzwerk verteilt werden und für Kollisionen ist nur der Zuständige Knoten verantwortlich.

1.4 Gruppierung nach Dimensionen

Die in dem vorherigen Schritt erstellte Tabelle, bestehend aus Signatur, Punkten und Dimensionen, wird nach Erstellen dieser, anhand der Spalte `Dimensionen` gruppiert. In diesem Schritt wäre es möglich, die Arbeit aufzuteilen und jeder Teilnehmer schaut in seinem zuständigen Tabellenbereich nach Überschneidungen und gruppiert diese. Anschließend muss jedoch das Ergebnis aller erneut auf Überschneidungen geprüft und gegebenenfalls erneut gruppiert werden. Je nach Masse der Daten könnte dies erneut verteilt werden, bevor eine Instanz die endgültige Berechnung vornimmt. Der Vorteil hierfür wäre, eine große Masse verteilt zu ermitteln und abschließend nur einen kleinen Teil für das Endergebnis berechnen zu müssen. Aber durch das erneute Prüfen der verteilt berechneten Zwischenergebnissen, entsteht mehr Aufwand als durch das einmalige iterieren über die Tabelle. Demzufolge muss überprüft werden, ob durch das Verteilen und wieder Zusammenfügen der Daten, wirklich ein Geschwindigkeitsvorteil gegenüber der direkten Berechnung auf einem Rechner entsteht oder ob dieser vernachlässigbar gering ist.

2 Framework und Architektur

2.1 Master-Worker-Architektur

Viele Algoritmusschritte von Subscale können teilweise vollständig parallelisiert werden. So is es möglich, Subscale über ein Datenset in Arbeitspakete aufzuteilen, die dann unabhängig bearbeitet werden können. Die Arbeitspakete werden in einer gemeinsamen Queue abgelegt. Zum System können sich beliebig viele Worker zuschalten. Die Worker verfügen über einen Client, der die Kommunikationstechnologie kennt, Arbeitspakete von der Warteschlange konsumieren und je nach Arbeitspakettyp intern routen kann. Für jeden Arbeitsschritt existiert eine eigene Anwendung. Die Anwendung konsumiert und entpackt das Arbeitspaket, erledigt die Aufgabe, verpackt das Resultat in einem neuen Arbeitspaket und platziert dieses wieder in der Queue. Zwischenergebnisse können unterstützend in einem geeignet schnellen Persistierungslayer gespeichert werden.

Vorteile:

- Das System lässt sich einfach auf beliebig viele Instanzen skalieren
- Der Algorithmus kann schrittweise parallelisiert werden
- Eine Benchmarkanalyse zur Performanzsteigerung ist durch den modularen Aufbau ohne großen Mehraufwand möglich
- Unterschiedliche Arbeitsschritte können mit unterschiedlichen und demzufolge den geeignetsten Technologien implementiert werden
- Der modulare Aufbau ermöglich eine konfliktarme parallele Implementierung

Nachteile:

- Es ist nicht bekannt, in wie weit die Queueing-Technologie oder der Persistierungslayer ein Bottleneck bildet
- Die Fehleranalyse erfordert ein umfangreiches Logging
- Asynchrone Abarbeitungen erhöhen die Aggregations- und Koordinationskomplexität

2.2 Pipeline

Um die im zuvor beschriebenen Kapitel möglichen Verteilungen zu realisieren, kann eine Pipeline aufgebaut werden, um diese Schritte abzuarbeiten. Dabei würde jeder Rechner Aufgaben aus einer Queue bekommen und diese an die Nächste Queue weiterschicken. Somit wäre eine gleichmäßige Verteilung der Aufgaben möglich und die Rechner sind

voneinander entkoppelt. Die Entkopplung der Rechner ermöglicht außerdem die Verwendung von verschiedenen Programmiersprachen bei den unterschiedlichen Aufgaben. Um dies zu Realisieren benötigt es jedoch einen Rechner der als Initiator des Algorithmus wirkt und einen Rechner zum Einsammeln der Ergebnisse, der am Ende der Pipeline auf die Berechnung wartet.

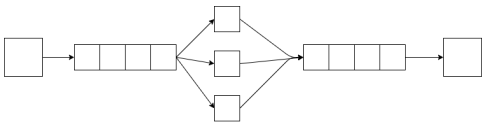

Abb. 1: Beispiel der Pipeline

2.3 Peer-to-Peer

Als weitere Möglichkeit besteht eine Peer-to-Peer Architektur. Der Vorteil dieser ist, dass eine sehr gute Skalierbarkeit vorhanden ist. Zusätzlich bietet diese eine gute Robustheit. Da jeder Peer sowohl Geber als auch Nehmer ist, kann eine Replikation und somit auch eine Fehlertoleranz ermöglicht werden. Der größte Nachteil ist jedoch die Komplexität, sowie der Aufwand der Sicherheit.

Als Erweiterung zu normalen Peer-to-Peer Systemen gibt es DistributedHashTabels (DHTs). Diese verbessern die Skalierbarkeit und besitzen in der Suche keine `false negatives`. Durch diese Besonderheiten, bietet sich eine DHT sehr gut an. Als einer der bekanntesten Algorithmen ist der CHORD bekannt, welcher die Hosts in einer Ringstruktur anordnet. Als Erweiterung zu diesem, werden sogenannte Fingertabellen hinzugefügt, welche Referenzen auf verschiedene Knoten haben. Ein Beispiel dieser Struktur ist in der folgenden Abbildung 2 zu sehen.

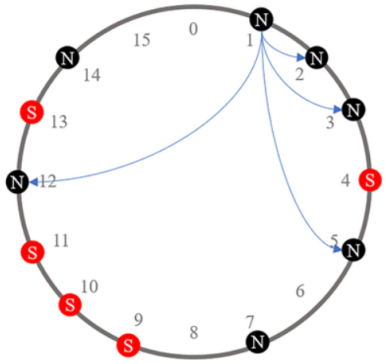

Abb. 2
Beispielhafte Darstellung eines Chordrings mit vier Fingern[5]

[5] https://www.researchgate.net/figure/Chord-Ring-mit-Fingertabelle-eines-Elements_fig1_327512285

Für die Implementierung ist man auf keine Programmiersprache oder ein bestimmtes Framework beschränkt. So kann über eine REST-Schnittstelle und einer Liste an Hosts, an welche sich gewendet werden kann, um beizutreten, die gesamte Struktur aufgebaut werden. Um ein Beispiel zu nennen, wäre Akka HTTP[6] eine gute Wahl für Java oder Scala.

2.4 TCP Posix

Mit Hilfe der Posix Bibliothek kann eine Client-Server Architektur aufgebaut werden, bei welcher sich die einzelnen Clients mit dem Server verbinden und der Server die Aufgaben verteilt und die Ergebnisse einsammelt.

Vorteile:

- Leichte Übertragbarkeit
- Es ist ein weitverbreiteter Standard, also existieren viele Dokumentationen und Beispiele im Internet

Nachteile:

- Es ist eine alte Technologie und auch komplexer als neuere Technologien

[6] https://doc.akka.io/docs/akka-http/current/index.html