https://www.infoq.com/news/2018/11/uber-big-data-evolution

Die Entwicklung von Ubers 100+ Petabyte Big Data Plattform

Ubers Ingenieurteam schrieb darüber, wie sich ihre große Datenplattform von traditionellen ETL-Aufträgen mit relationalen Datenbanken zu einer auf Basis von Hadoop und Spark entwickelte. Ein skalierbares Aufnahmemodell, ein Standard-Transferformat und eine benutzerdefinierte Bibliothek für inkrementelle Updates sind die Schlüsselkomponenten der Plattform.

Verschiedene Teams bei Uber verwenden große Datenmengen für Dinge wie die Vorhersage der Fahrtennachfrage, Betrugserkennung, geografische Berechnung und die Behebung von Engpässen im Anmeldeprozess des Mitfahrers. Ihre erste Lösung, die vor 2014 entwickelt wurde, basierte auf MySQL und PostgreSQL. Die damals relativ geringe Datenmenge - ein paar TB (Terabyte)- konnte in diese RDBMSs passen, und die Benutzer mussten selbst herausfinden, wie man übergreifend Datenbanken abfragt. Die Stadtbetriebsteams, Datenwissenschaftler und Analysten sowie die Ingenieurteams nutzten diese Daten. Die Bemühungen um Standardisierung führten zur Einführung von Vertica - einer spaltenorientierten Analyseplattform - unterstützt durch Ad-hoc-Extrakt-Transform-Load (ETL)-Aufträge. Ein benutzerdefinierter Abfrageservice ermöglichte den Zugriff auf die Daten über SQL. Die Datenmenge wuchs auf 10 TBs, begleitet von einem Wachstum der Anzahl der Teams und Dienste, die diese Daten verwendeten. Die Hauptprobleme, mit denen Uber in dieser Phase konfrontiert war, waren mangelnde horizontale Skalierbarkeit, steigende Kosten und Datenverlust, die sich aus dem Fehlen eines formalen Schemas zwischen Datenproduzenten und Verbrauchern ergaben.

Das Engineering-Team hat Hadoop in der nächsten Phase eingesetzt, um Daten aus mehreren Filialen zu übernehmen, ohne sie zu transformieren. Apache Spark, Apache Hive und Presto als Abfragemaschine waren Teil der Zusammenstellung. Vertica war schnell, konnte aber nicht billig skalieren, während Hive das umgekehrte Problem hatte (PDF). Die gemeinsame Speicherung des Schemas und der Daten über einen eigenen Schema-Service löste die Probleme der vorherigen Phase. Die Datenmenge wuchs auf 10s PBs, und die Dateninfrastruktur führte 100k Jobs pro Tag über 10000 virtuelle CPU-Kerne aus.

Trotz der horizontalen Skalierbarkeit stießen sie auf HDFS-Engpässe. In einem HDFS-Cluster verfolgt ein NameNode, wo jede Datei im Cluster aufbewahrt wird, und verwaltet den Verzeichnisbaum. HDFS ist für den Streamingzugriff auf große Dateien optimiert, und viele kleine Dateien machen den Zugriff ineffizient. Uber stieß auf dieses Problem, als ihr Datenvolumen über 10 PB hinausging. Sie umgingen die HDFS-Engpässe, indem sie die NameNode Garbage Collection optimierten, die Anzahl der kleinen Dateien begrenzten und einen HDFS-Lastmanagementdienst einrichteten. Darüber hinaus waren die Daten für

den Endanwender nicht schnell genug verfügbar. Reza Shiftehfar, Engineering Manager bei Uber, schreibt, dass „Ubers Geschäft in Echtzeit läuft und als solches benötigen unsere Dienstleistungen Zugang zu so frischen Daten wie möglich. Um die Datenbereitstellung zu beschleunigen, mussten wir unsere Pipeline auf die schrittweise Aufnahme von nur aktualisierten und neuen Daten umstellen."

Abb. 1

Das Ergebnis war eine eigene Spark(Zünd)-Bibliothek namens Hudi (Hadoop Upserts andD Incrementals (zuwachsbezogen)). Es bildet eine Schicht auf HDFS und Parkett (ein Speicherdateifo rmat), die Aktualisierungen und Löschungen ermöglicht und damit das Ziel erreicht, dass ETL-Aufträge inkrementell werden. Hudi arbeitet damit, dass Benutzer nach dem letzten Checkpoint Zeitstempel abfragen können, um alle Daten abzurufen, die seit dem Checkpoint aktualisiert wurden, ohne dass eine vollständige Tabellenprüfung durchgeführt werden muss. Dadurch sank die Latenzzeit von 24 Stunden auf weniger als eine Stunde für modellierte Daten und 30 Minuten für Rohdaten.

Neben Hudi ist die andere Ergänzung zur neuesten Phase der großen Datenplattform von Uber die Datenaufnahme durch Apache Kafka mit angehängten Metadaten-Headern. Eine Komponente namens Marmaray nimmt die Änderungen von Kafka auf und schiebt sie über die Hudi-Bibliothek nach Hadoop. All dies wird mit Apache Mesos und YARN orchestriert. Während Mesos für lang laufende Dienste geeignet ist, ist YARN besser für Batch(Stapel)/Hadoop-Jobs geeignet. Uber verwendet sein benutzerdefiniertes Terminierungs-Programmiergerüst Peloton, das auf Mesos aufbaut, um sein Rechner-Arbeitsaufkommen zu verwalten.