entwickler.press
shortcuts

Big Data

Apache Hadoop

Bernd Fondermann, Kai Spichale,
Lars George

Bernd Fondermann, Kai Spichale, Lars George

Big Data

Apache Hadoop

ISBN: 978-3-86802-400-5

Ein Imprint der Software & Support Media GmbH

1 Daten im großen Stil – Apache Hadoop

von Bernd Fondermann

Doug Cutting hatte ein Problem, für das Internetarchiv das Internet (sprich alles HTML) herunterzuladen und zu speichern. Das war vor zehn Jahren schon eine Herausforderung, heute ist sie mit dem exponentiellen Anwachsen der Daten nicht kleiner. Allein die schiere Datenmenge: Petabyte, also Millionen von Gigabyte.

Und selbstverständlich kann man diese Daten in einer riesigen, über das Netzwerk erreichbaren Festplatte ablegen, einem Network Attached Storage (NAS) wie sie heute verfügbar sind. Solche Lösungen sind aber nicht nur teuer, sie haben auch einen entscheidenden architektonischen Nachteil: Zum massiven Verarbeiten der Files müssen sie über das Netzwerk zum entsprechenden Programm geschleust werden. Was die Laufzeit substanziell verlängern kann. Soll also stattdessen alles in Scharen von relationalen Datenbanken untergebracht werden? Clustering von RDB ist auch heute noch kein Mainstream. Zudem sind Oracle, Postgres und Co. exzellent geeignet für normalisierte Daten. Das ist eine Eigenschaft, die Quellcode von Webseiten nicht hat. Für viele nebenläufige, beliebige Lesezugriffe sind relationale DBs zwar geeignet, aber nicht optimiert. Und wehe, es kommen ein paar Schreiboperation dazwischen. Außerdem sind sie extrem gut darin, einen konsistenten Zustand über den gesamten Bestand zu gewährleisten und strukturierte Daten miteinander zu verknüpfen. Das ist entscheidend für das E-Business, in dessen Aufschwungphase relationale Datenbanken groß geworden sind. Für die Verarbeitung von gecrawlten Webseiten sind andere Dinge viel wichtiger: Redundanz, Verteilung, Durchsatz,

Skalierbarkeit auf große Datenmengen, Toleranz gegenüber Ausfällen von Teilsystemen.

Der Batch-Job, das hässliche Entlein?

Langlaufende Batch-Jobs besuchen die Websites, speichern sie historisiert ab und verarbeiten sie weiter. Es wurde Cutting schnell klar, dass die klassische Batch-Verarbeitung hier nicht ausreicht [1]. Doch wie schafft man es, dass solche Prozesse weiterlaufen, auch wenn Teile Fehler generieren, und wie kann man sie effektiv und effizient über möglichst viele Maschinen verteilen, auch wenn man nicht vorhersagen kann, wo Daten zusammenhängen und eigentlich gemeinsam verarbeitet werden müssen? Cutting stieß im Internet auf ein Paper einer Firma namens Google Inc., die dasselbe Problem hatte und eine Lösung dafür vorstellte: MapReduce beschreibt eine verteilte Ablaufumgebung, die grob gesagt in zwei Schritten aus Inputdaten neue Daten generiert. Dabei ist die Struktur von Input und Output unerheblich. Die Idee ist so einfach wie genial, aber für relational Geschulte (und wer könnte sich davon ausnehmen) fremd: In einem ersten Schritt, *Map* genannt, werden alle Datensätze in verteilten Prozessen gelesen, verarbeitet und die Ergebnisse unabhängig voneinander (parallelisiert) in eine Key-Value-Datenstruktur (eine Multi-Map) geschrieben. Dabei achtet man darauf, dass genau *die* Daten denselben Key erhalten, die im zweiten, dem *Reduce*-Schritt, *gemeinsam* weiterverarbeitet werden. Werte zu *unterschiedlichen* Keys werden unabhängig voneinander gegebenenfalls parallel weiterverarbeitet. So erreicht man, dass beide Phasen hochparallel und verteilt ablaufen und dennoch Zusammenhänge innerhalb von großen Datenmengen hergestellt werden können. Cutting implementierte MapReduce Mitte des vergangenen Jahrzehnts als Open Source in Java und nannte das Projekt Hadoop [2]. Kai Spichale beschäftigt sich in einem eigenen Abschnitt genauer mit MapReduce (siehe „2 - Hadoop MapReduce“).