Lambda-Architektur und Hadoop Framework in der Praxis

Big Data **für Macher**

Fragt man einen Informatikprofessor nach Big Data, bekommt man als Antwort meist eine Ausführung über die 3 „Vs" (Variety, Velocity und Volume) und die damit verbundenen Anforderungen an Big-Data-Systeme. Ein Datenbankadministrator hingegen denkt bei Big Data an Partitionierung, Sharding und seine Oracle Exadata, die gewaltige Datenmengen zum Abruf bereithält.

Die meisten Nicht-Techniker assoziieren mit Big Data in erster Linie den gläsernen Menschen sowie Google und andere Unternehmen, die die Menschen mittlerweile besser kennen als sie sich selbst. Und dann gibt es noch die Zyniker, für die das ganze Thema nur der nächste große Hype ist: Die Verarbeitung großer Datenmengen gab es schließlich schon in den 70ern, nur gab es damals noch keine Tools mit farbigen Tierbildchen als Logo. Wie aber blicken IT-Architekten als die eigentlichen Macher auf Big Data? Und: Wie sieht eine Big-Data-Architektur in der Praxis aus? Dieser Artikel stellt ein Anwendungsbeispiel vor und erläutert das Architekturkonzept und die Entscheidung für bestimmte Tools im Detail.

WEB-TIPP:
www.opitz-consulting.com

Das Hadoop Ökosystem

Für jemanden, der sich nur einen schnellen technischen Überblick über das Thema verschaffen möchte, ist Big Data ein Alptraum. Auf ihn warten hunderte von Tools mit bunten Logos, im Hadoop-Kontext gerne nach Tieren benannt, teilweise mit sehr speziellen Einsatzzwecken.

Dabei besteht Hadoop im Wesentlichen nur aus zwei Komponenten: Zum einen aus dem Ressource Manager YARN, zum anderen aus dem verteilten Dateisystem Hadoop File System (HDFS). Sämtliche Erweiterungen, im Folgenden „Hadoop Tools" genannt, basieren auf einer dieser Komponenten oder auf beiden. Vor allem zu Beginn eines Big-Data-Vorhabens muss der IT-Architekt die Tools und ihr Zusammenspiel gut kennen, um eine sinnvolle Auswahl treffen zu können.

Die Lambda-Architektur

Nach Big-Data-Architekturen gefragt, fällt im Architekturumfeld meist schnell der Begriff Lambda-Architektur. Die Umsetzung einer solchen Architektur im Hadoop Ökosystem erweist sich dabei als komplexes Zusammenspiel verschiedener Tools.

Die Lambda-Architektur beschreibt den konzeptionellen Aufbau einer Big-Data-Architektur bestehend aus einem zentralen Eintrittspunkt in das Big-Data-System, dem Ingestion-Layer, einem Speed- und einem Batch-Verarbeitungslayer sowie dem Serving Layer (vgl. Bild 1). In der Big-Data-Welt gibt es eine Vielzahl an Tools, um eine solche Architektur basierend auf Open-Source-Technologien zu entwickeln.

Für die Datenstrom-Verarbeitung ist die Lambda-Architektur das bekannteste Design Pattern im Big-Data-Umfeld. In vielen Fällen wird die Lambda-Architektur nicht in ihrer reinen Form umgesetzt, sondern in einer Mischform. In einer solchen Ausprägung sieht die Systemübersicht zwar aus wie in einer Lambda-Architektur, das Zusammenspiel der Komponenten ist allerdings anders – und das kann gute Gründe haben. Bild 1 zeigt den Aufbau dieses Architektur-Konzepts.

In der Lambda-Architektur kommen Datenströme zunächst im Data Ingestion Layer an. Der Layer ist dafür zuständig, die ankommenden Daten temporär zu puffern und diese den beiden nachfolgenden Layern, dem Batch Layer und

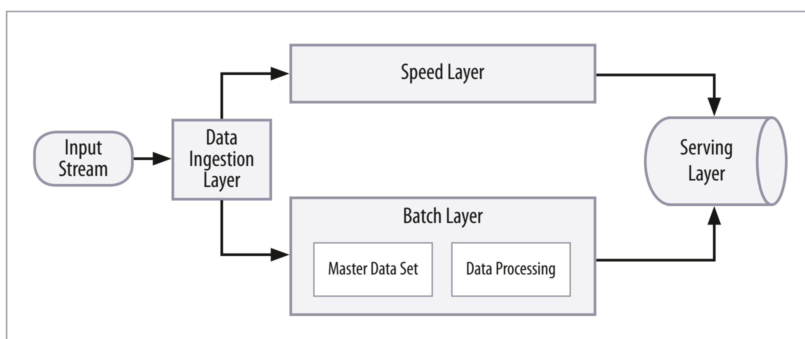


Bild 1: Lambda-Architektur.

dem Speed Layer bereitzustellen. Der Batch Layer besteht aus einem Master Data Set und der Verarbeitungslogik. Im Master Data Set werden alle Daten, die den Batch Layer erreichen, gespeichert. In bestimmten Intervallen, zum Beispiel einmal am Tag, wird die Batch-Verarbeitung gestartet und damit die Verarbeitungslogik auf die Daten im Master Data Set angewendet. Die Ergebnisse dieses Prozesses werden im Serving Layer gespeichert.

Diese Ergebnisse nennen wir „Intermediate View". Warum „intermediate"? Die Daten werden nur in bestimmten Intervallen, zum Beispiel alle 24 Stunden, aktualisiert. Zwischen zwei Batch-Durchläufen strömen aber weiterhin Daten auf das System ein. Hier kommt der Speed Layer ins Spiel. In der Lambda-Architektur erledigt der Speed Layer die gleichen Aufgaben wie der Batch Layer, allerdings in Real-Time. Auch er speichert die Ergebnisse im Serving Layer. Diese Ergebnisse nennen wir „Streaming View". Fassen wir die Daten der „Intermediate View" und der „Streaming View" zu einem beliebigen Zeitpunkt zu einer „Aggregated View" zusammen, erhalten wir eine Live-Sicht auf die neuesten Berechnungen des Speed Layers und auf die älteren Berechnungen des Batch Layers.

Der Speed Layer kümmert sich somit um die Zwischenverarbeitung der Daten, bis der Batch Layer die Verarbeitungslogik das nächste Mal anstößt. Sein Ziel ist es, die permanente Aktualität aller Daten im Serving Layer sicherzustellen, eine Leistung, die der Batch Layer alleine nicht erfüllen kann. Im Internet finden sich aber auch weniger striktere Auffassungen zur Lambda-Architektur. So nennen einige Entwickler ein System bereits „Lambda-Architektur", wenn der Speed Layer für ganz andere Aufgaben als der Batch Layer zuständig ist.

Wie aber lassen sich die oben erwähnten Tools und die Lambda-Architektur zu einer Grobarchitektur kombinieren?

Der Use Case

Im Anwendungsfall eines Automatenaufstellers haben wir vor einiger Zeit eine Lambda-Architektur implementiert. Der Use Case ist folgender: Snack-Automaten, wie sie häufig an Bahnhöfen oder in Unternehmen zu finden sind, stehen mit dem Backend des Aufstellers in Verbindung. Jeder Verkauf eines Produkts aber auch Daten über den Wechselgeldzustand oder Sensorwerte, wie sie zum Beispiel die Temperatursteuerung ausgibt, werden an dieses Backend übertragen. Hier werden alle Daten gespeichert und nächtliche Auswertungen zum Beispiel über Aggregationen von Verkäufen nach Verkaufsregionen oder Tageszeiten durchgeführt.

Zusätzlich zu den nächtlichen Auswertungen erfolgt eine Real-Time-Auswertung der Daten. So kann das Personal bei stark steigenden Umsatz an einem Automaten oder in einer Region schnell reagieren und zum Beispiel einen Zigarettenautomaten während eines Volksfestes oder einen Snack-Automaten am Bahnhof bei einem Fußballspiel rechtzeitig auffüllen. Auf der anderen Seite sollen Sensorwerte, wie die genannten Daten der Temperatursteuerung, ständig beobachtet werden. Mit ihrer Hilfe kann das Unternehmen zum Beispiel Spuren von Vandalismus sofort entdecken.

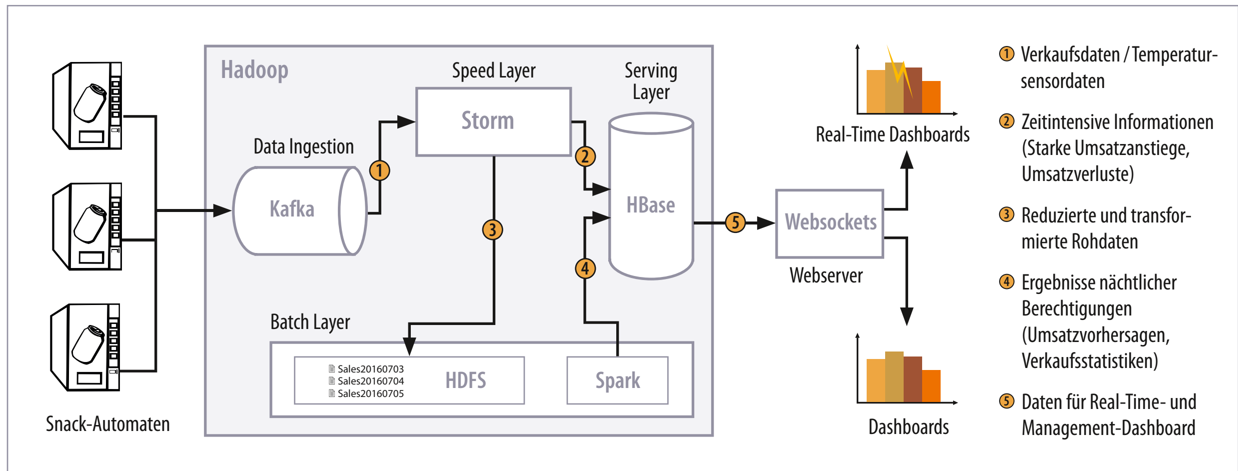


Bild 2: An die Lambda-Architektur angelehnte Grobarchitektur.

Ziel ist letztendlich, eine automatisierte Einsatzplanung für das Auffüllen und die technische Wartung zu entwerfen, die die zuständigen Mitarbeiter anhand der gesammelten Daten live auf ihrem Smartphone einsehen können. In einer späteren Ausbaustufe erfolgt eine automatisierte Produktplanung, die vorgeben kann, mit welchen Produkten die Automaten zu welcher Zeit und in welcher Menge bestückt werden sollten, um den Konsumenten immer das passende Sortiment anzubieten. An diesen Punkt gelangt das System unter anderem durch den Einsatz von Machine-Learning-Technologien.

Die Umsetzung

Über die Anforderungen und das Konzept der Lambda-Architektur sei genug gesagt. Nun soll es an die praktische Umsetzung gehen. Bild 2 zeigt die Grobarchitektur, die wir für unseren Use Case gewählt haben. Wie aber gestaltete sich die Tool-Auswahl und wie haben wir die gewählten Tools eingesetzt?

Als Basis für das Projekt fiel unsere Wahl auf Apache Hadoop, das De-Facto-Standard-Framework für skalierbare und verteilt arbeitende On-Premises-Software. Bei Hadoop selbst handelt es sich zwar um eine Open-Source-Technologie. Doch in der Praxis werden fast immer Distributionen von Herstellern verwendet. Diese haben einige Vorteile vorzuweisen, wie eine vereinfachte Installation, Support und ein größeres Toolset. Bekannte Hersteller von Hadoop Distributionen sind Cloudera, MapR und Hortonworks. Alle Hersteller liefern die wichtigsten Tools in ihren Distributionen aus.

In unserem Anwendungsfall haben wir die Hortonworks Data Platform verwendet. Es hätte aber genauso gut das Produkt eines der anderen großen Hersteller zum Einsatz kommen können.

Als Data Ingestion System kam Apache Kafka zum Einsatz. Kafka bietet, wie alle Tools im Hadoop Ökosystem, von Haus aus die Möglichkeit, linear zu skalieren. Dabei stellt Kafka kein System für die langfristige Persistierung von Daten dar, sondern eher ein Queuing System. Durch das Konzept der Consumer Groups können Daten sowohl wie in einer klassischen Queue verarbeitet werden, bei der Consumer jede Nachricht exakt einmal lesen, als auch wie in einem Publish-Subscribe-System, in dem sich beliebig viele Consumer auf alle Daten registrieren können. Kafka unterstützt dabei nicht den JMS-Standard und ist im Vergleich mit klassischen JMS-Systemen in seiner Funktionalität deutlich abgespeckter. In Sachen Datendurchsatz ist Kafka allen anderen Lösungen allerdings haushoch überlegen.

Den Speed Layer haben wir mit dem „Dinosaurier" des Stream Processings, Apache Storm, realisiert. Alternativen gäbe es mit Flink, Spark Streaming und Kafka Streams genug. Storm bietet im Vergleich zu den anderen Technologien einen eher mittelmäßigen Datendurchsatz, dafür aber eine sehr hohe Performance, was für unseren Anwendungsfall perfekt war. Auch bietet Storm eine eher geringe Einarbeitungszeit. Storm Jobs können quasi in beliebigen Programmiersprachen geschrieben werden und auch das Schreiben von funktionalem Programmier-Code ist keine Voraussetzung wie bei manchen anderen Tools. Die errechneten Daten, wie die durchschnittlichen Verkaufszahlen eines Automaten über ein Zeitfenster von einer Stunde, speichert das System im Serving Layer.

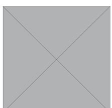
Im Gegensatz zur klassischen Definition einer Lambda-Architektur werden die Daten für den Batch-Layer auch hier noch einmal zusätzlich im Speed

„Mithilfe des Lambda-Architektur-Konzepts haben wir eine Architektur geschaffen, für die immer stärker wachsende Datenberge und neue Anforderungen der Zukunft kein Problem darstellen."

Lukas Berle, Big Data Software Engineering Lead bei Opitz Consulting

Weiterführende Informationen: www.it-daily.net

Whitepaper



Layer verarbeitet. Das liegt daran, dass die Daten vor der Speicherung noch transformiert werden müssen. Mittlerweile bietet Kafka mit Kafka Streams die Möglichkeit, solche Transformationen direkt in Kafka zu verarbeiten, dieses Feature stand uns aber noch nicht zur Verfügung. Man sieht an diesem Beispiel, wie schnell die Entwicklung der Big-Data-Tools voranschreitet.

Im Batch Layer werden die Daten im Hadoop Distributed File System (HDFS) gespeichert. HDFS ist hervorragend geeignet, um Daten redundant und dank eingebauter Server-Rack Awareness sicher zu speichern. HDFS ist außerdem darauf ausgelegt, sehr große Datenmengen nach außen zugreifbar zu machen. Seine Grenzen erfährt es allerdings bei wahlfreien Zugriffen und hohen Anforderungen an Latenzzeiten.

Die Verarbeitung

Zur Verarbeitung der Daten aus dem HDFS fiel unsere Wahl auf Apache Spark. Spark liest die Daten alle 24 Stunden aus dem HDFS und verarbeitet sie erneut. Dabei werden die Berechnungen des Speed Layers erneut durchgeführt. Warum das? Bestimmte Daten könnten den Speed Layer stark zeitversetzt erreicht haben. So könnten aufgrund einer Netzstörung an einem Automaten bestimmte Verkaufsdaten unser Hadoop System erst einen Tag später erreichen. Anhand des Datumsstempels erkennt der Speed Layer das Entstehungsdatum des entsprechenden Datensatzes und verwirft ihn, da er nur Daten der letzten Stunde im Speicher hält. Mit einem älteren Datum weiß er nichts anzufangen. Spark führt noch weitere Berechnungen durch, zum Beispiel für Nachfüllprognosen, perspektivisch kommen auch noch Berechnungen aus dem Bereich des Machine Learnings hinzu.

Auch die Ergebnisse des Batch Layers werden im Serving Layer gespeichert. In unserem Beispiel haben wir dies mit Apache HBase realisiert. HBase selbst ist bereits Bestandteil der meisten Hadoop Installationen und gehört innerhalb der Typisierung von NoSQL-Datenbanken zu den sogenannten Column Family Stores. HBase ist hervorragend für die schnelle Speicherung und Abfrage von Daten geeignet und wie die meisten NoSQL-Datenbanken wegen ihrer Fähigkeit zur linearen Skalierung für Anwendungszwecke im Big-Data-Bereich prädestiniert. Zur Anzeige werden die Daten aus HBase schließlich durch einen Webserver ausgelesen, der ein Real-Time Dashboard und ein klassisches Management Dashboard mit Daten versorgt.

Die beschriebene Architektur lässt erahnen, wie viele verschiedene Tools für die Realisierung eines Big-Data-Systems nötig sind. Komplexere Anwendungsfälle erfordern natürlich ein noch größeres Toolset, das auch Themen wie Security, Berechtigungskonzepte und Workflowmanagement umfasst. Bleibt abzuwarten, wie weit sich die verschiedenen Tools noch entwickeln. Spark hat sich mittlerweile als De-Facto-Standard der Batch-Verarbeitung durchgesetzt, Kafka beginnt mit Kafka Streams in den Bereich des In-Memory Stream Processings vorzustoßen und auch Spark Streaming ist eine ernstzunehmende Alternative zu klassischen Streaming Frameworks geworden.

Fazit

Wie geht es mit unserem Beispielprojekt weiter? Für die langfristige Wartbarkeit werden wir DevOps-Strategien entwickeln, zum Beispiel für automatisiertes Cluster-Setup, Konfiguration und Deployment. Eine Sache würden wir bei zukünftigen Projekten übrigens anders machen: Um doppelte Programm-Logik in Speed Layer und Batch-Layer zu vermeiden, würden wir im Nachhinein auch Spark Streaming für den Speed-Layer in die engere Wahl ziehen.

Insgesamt sind wir mit dem Verlauf dieses Big-Data-Projekts sehr zufrieden: Statt einer teuren Spezialsoftware, die nur aktuelle Anforderungen abdeckt, steht dem Kunden nun eine Plattform zu Verfügung, die Einsatzzwecke von Datenspeicherung und Datenverarbeitung hochskalierbar abdeckt. Mithilfe des Lambda-Architektur-Konzepts haben wir eine Architektur geschaffen, für die immer stärker wachsende Datenberge und neue Anforderungen der Zukunft kein Problem darstellen.

Lukas Berle

IMPRESSUM