Streamingarchitekturen in der Praxis

Lambda vs. Kappa

Um den Anschluss an die Digitalisierung nicht zu verlieren, wird es Zeit, sich Gedanken über eine geeignete, zukunftssichere Datenarchitektur zu machen. Doch gibt es sie, die eine Streamingarchitektur? Die Praxis zeigt: Im Einzelfall erfolgt eine Auswahl aus vielen Einzelkomponenten.

AUTOR: LUKAS BERLE

Neue, disruptive Geschäftsmodelle basieren oft auf den Faktoren Daten, Geschwindigkeit und Skalierung. Während Fachabteilungen zusammen mit der IT an neuen Ideen arbeiten, um in Zeiten zunehmender Digitalisierung nicht abgehängt zu werden, müssen auch aus technologischer Sicht neue Wege gegangen werden. Technologieverantwortliche im Unternehmen benötigen eine leistungsstarke, zukunftssichere Datenarchitektur. Ein Teil von ihr kann die Auswertung von hochfrequenten Datenströmen abdecken. Viele Unternehmen sehen ein besonders großes Potenzial in der Nutzung von operativen Daten. Also beispielsweise von Sensordaten von Maschinen, von Serverlogdaten oder von Eventdaten, wie Verkaufsinformationen oder Positionsdaten einer Fahrzeugflotte.

Aus Businessperspektive haben diese Daten eine besondere Gemeinsamkeit: Ihr Wert nimmt sofort nach dem Ereigniszeitpunkt stark ab (**Abb. 1**). Wird nicht innerhalb von Sekunden oder Sekundenbruchteilen auf sie reagiert, haben diese Daten oft kaum noch einen Wert für das Unternehmen. Denn die Sensordaten einer Maschine bringen nichts, wenn wir sie erst verarbeiten, nachdem die Maschine schon überhitzt ausgefallen ist, obwohl wir dies bei sofortiger Analyse hätten vorhersehen können. Auch maßgeschneiderte Angebote für einen Webseitenbesucher bringen uns nichts, wenn dieser unsere Seite schon längst wieder verlassen hat.

Mit der Nutzung von operativen Daten gewinnt also auch die schnelle Auswertung dieser Daten an Bedeutung. Zwar haben viele IT-Abteilungen jahrelange Erfahrung in der Entwicklung von klassischen Enterprise-Lösungen, die Entwicklung von In-Memory-Stream-Processing-Lösungen fordert allerdings Antworten auf neue Probleme: Extrem hohe Ausfallsicherheit, beliebige Skalierbarkeit und Verarbeitungsgarantien sind nur einige Punkte, denen besondere Beachtung geschenkt werden muss.

Für einen Automatenhersteller haben wir einen Prototyp umgesetzt, der Snackautomaten, z. B. an Bahnhöfen, an ein zentrales IoT-System anbindet. Jedes Verkaufsevent, aber auch Sensordaten wie Ergebnisse der Temperatursteuerung, sollte an das System gesendet werden. Damit ermittelt das System, wann die Produkte das nächste Mal nachgefüllt werden müssen. Denn bei kurzfristig stark wachsenden Umsätzen an einem Automaten, z. B. bei Fußballspielen oder während Volksfesten, müssen Automaten schnellstmöglich nachgefüllt werden. Die Einsatzplanung der zuständigen Mitarbeiter sollte so künftig

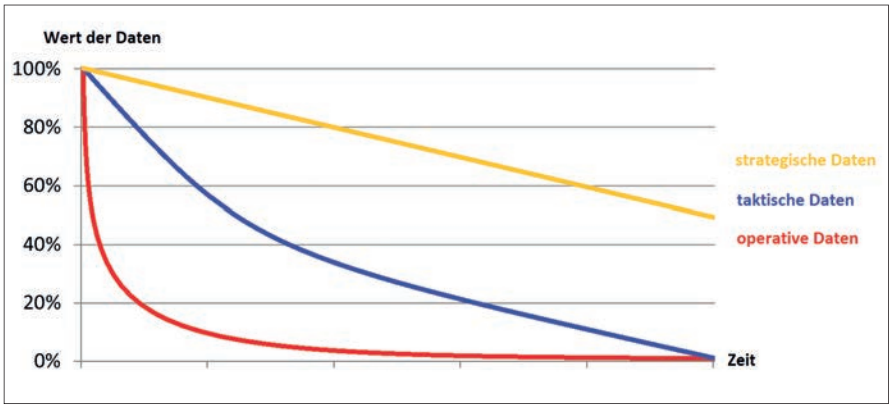

Abb. 1: Der Wert von Daten nimmt über die Zeit ab (in Anlehnung an [1])

Abb. 2: Ein Auszug populärer Tools aus dem Big-Data-Ökosystem

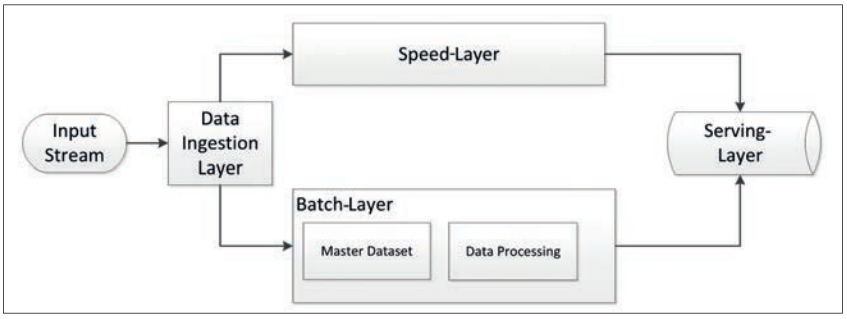

Abb. 3: Die Lambdaarchitektur

vollständig automatisiert erfolgen. Auch die Fakten, die sich auf den Verkauf auswirken, sollten zeitnah ermittelt werden, z. B., dass Produkte an bestimmten Tagen oder zu bestimmten Jahreszeiten besonders oft verkauft werden. So kann der Anbieter in der Sortimentsplanung schnell reagieren. Durch die ständige Analyse der Sensordaten, z. B. der Daten der Temperatursteuerung, kann der Anbieter Ereignisse wie technische Defekte oder Vandalismus schnell entdecken und, wo nötig, einen Techniker in den Einsatz schicken. Perspektivisch sollen mithilfe der Methoden des maschinellen Lernens langfristige Absatzprognosen für das Käuferklientel erstellt werden.

TOOLS UND ARCHITEKTUREN FÜR STREAMINGDATEN

Das Big-Data-Ökosystem bietet für eine Vielzahl dieser Anwendungsfälle technische Lösungen. Der Einstieg fällt vielen IT-Architekten überraschend schwer: Rund um Hadoop gibt es unzählige Tools und Frameworks, die Einsatzzwecke wie Batch Processing, Stream Processing, Data Ingestion, Security, Orchestrierung und vieles mehr abdecken. **Abbildung 2** zeigt einige populäre Tools aus dem Big-Data-Umfeld.

Gerade im Bereich Stream Processing wurde in den letzten Jahren eine bemerkenswerte Anzahl an neuen Tools auf den Markt gebracht. Allein im Open-Source-Bereich gibt es eine zweistellige Anzahl an Streamingtechnologien, größtenteils betreut von der Apache Software Foundation. Der volle Wert dieser Technologien entfaltet sich oft erst, wenn sie als Teil einer Streamingarchitektur zum Einsatz kommen. Im Big-Data-Bereich sind dabei vor allem zwei Architekturpatterns bekannt: Die Lambdaarchitektur und die Kappa-Architektur. Die Lambdaarchitektur ist historisch gesehen älter, daher wollen wir sie hier als Erstes zum Zug kommen lassen.

DIE LAMBDAARCHITEKTUR

Die Lambdaarchitektur (**Abb. 3**) wurde 2011 von Nathan Marz vorgestellt. Es heißt, der Name sei auf die Ähnlichkeit der grafischen Darstellung der Architektur mit dem Buchstaben Lambda zurückzuführen, der in diesem Fall nach links gedreht wurde. Andere vermuten den Ursprung des Namens in den Lambdafunktionen, die als eine der Grundlagen der Funktionalen Programmierung gelten. Die Lambdaarchitektur besteht aus einem zentralen Eintrittspunkt in das Datenverarbeitungssystem, dem Data-Ingestion-Layer. Von diesem ausgehend werden Daten sowohl zum Batch- als auch zum Speed-Layer gesendet. Der Batch-Layer speichert die einlaufenden Daten zunächst in einem Master Dataset. Nach einem fest definierten Intervall, z. B. einmal am Tag, wird der Batch Job gestartet und die bis dahin im Master Dataset aufgelaufenen Daten werden verarbeitet. Die errechneten Ergebnisse werden im Serving-Layer gespeichert. Damit ist der Serving-Layer nach Abschluss des Batch Jobs direkt aktuell. Da Datenströme durchgängig auf unser System einströmen und somit schnell an Aktualität verlieren, gibt es den Speed-Layer. Er erhält ebenfalls alle Daten, persistiert diese aber nicht auf der Festplatte. Im Gegenteil: Die Daten werden direkt – meist in-memory – verarbeitet und die Ergebnisse ebenfalls im Serving-Layer gespeichert. Damit befinden sich im Serving-Layer immer die jeweils neuesten, in Echtzeit verarbeiteten Daten.

DIE KAPPA-ARCHITEKTUR

Das Pendant zur Lambdaarchitektur ist die Kappa-Architektur (**Abb. 4**). Entworfen wurde diese von Jay Kreps, dem Initiator bekannter Big-Data-Technologi-

en wie Kafka und Samza. Die Grundüberlegung zur Kappa-Architektur ist einfach erklärt. In der Lambdaarchitektur haben wir Batch- und Speed-Layer, wobei der Speed-Layer dafür zuständig ist, den Serving-Layer immer auf dem aktuellen Stand zu halten. Da stellte sich für Kreps die berechtigte Frage: Brauchen wir überhaupt einen Batch-Layer? Die Kappa-Architektur besteht tatsächlich nur aus einem Data-Ingestion-Layer und einem Speed-Layer sowie dem Master Dataset. Alle Daten, die in das System eintreten, gehen also durch den Speed-Layer. Die errechneten Ergebnisse werden im Serving-Layer gespeichert. Dies hat gegenüber der Lambdaarchitektur Vor- und Nachteile. Ein Vorteil ist sicherlich, dass keine unterschiedlichen Programme mehr laufen, die separat entwickelt und gepflegt werden müssen. Gleichzeitig stellt sich die Frage, was passieren soll, wenn sich die Programmlogik des Speed-Layers in der Kappa-Architektur ändert. Die bisherigen im Speed-Layer gespeicherten Daten wurden schließlich mit der vorherigen Programmversion errechnet und müssen aktualisiert werden. In der Lambdaarchitektur könnte dies der Batch-Layer im nächsten nächtlichen Durchlauf erledigen. In der Kappa-Architektur hingegen werden in solch einem Fall sämtliche im Master Dataset gespeicherte Daten noch einmal durch den Speed-Layer verarbeitet, um die Daten im Serving-Layer zu aktualisieren.

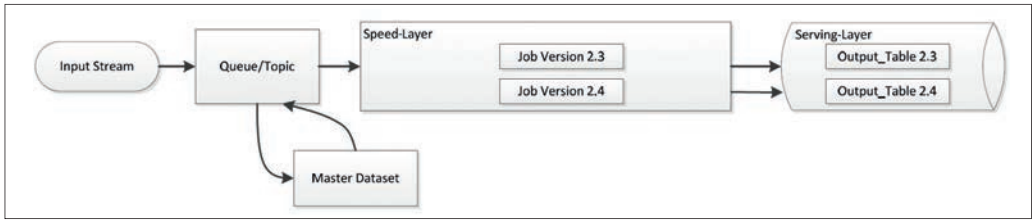

Abb. 4: Die Kappa-Architektur

UNSERE ENTSCHEIDUNG: LAMBDA

Für unseren Prototyp haben wir uns letztendlich für die Lambdaarchitektur entschieden. Denn die Grundvoraussetzung für die Kappa-Architektur, dass sich die Anwendungsfälle immer als ein Streamingproblem abdecken lassen, konnte hier nicht immer erfüllt werden. Für einige Fälle, wie das Erstellen der Absatzprognosen, hätten wir auf weit zurückliegende Daten zugreifen müssen, um verschiedene Kennzahlen zu bilden. Es wäre unpraktikabel, diese Daten eine solch lange Zeit im Speed-Layer zu halten, wie es die Kappa-Architektur erfordert. Auch sollten diese Daten die Grundlage für das Training von Machine-Learning-Modellen bilden. Viele dieser Modelle lassen sich in einem Speed-Layer nicht inkrementell mit neuen Daten erstellen, denn sie benötigen einen von vornherein vorhandenen Datensatz. Kurz: Ein Batch-Verarbeitungslayer wird benötigt. Auch andere technische Gründe sprachen in unserem Fall für die Lambdaarchitektur. Die konkrete Umsetzung der Architektur inklusive der eingesetzten Tools ist in **Abbildung 5** dargestellt.

Für das Data-Ingestion-System fiel die Wahl auf Apache Kafka. Kafka ist der De-facto-Standard für Messagingsysteme im datengetriebenen Umfeld, und dank seiner guten Skalierbarkeit und seiner überragenden Performance war es für uns die perfekte Lösung. Für den Batch-Layer war klar: Hadoops MapReduce-Implementierung ist alles andere als State of the Art. Die am weitesten verbreitete Technologie in der Batch-Verarbeitung ist definitiv Apache Spark. Auch ist Spark Bestandteil aller bekannteren Hadoop-Distributionen von Hortonworks oder Cloudera. Würden wir die Entscheidung heute erneut treffen, würden wir übrigens noch andere in der Zwischenzeit gereifte Technologien stärker in der Vorauswahl berücksich-

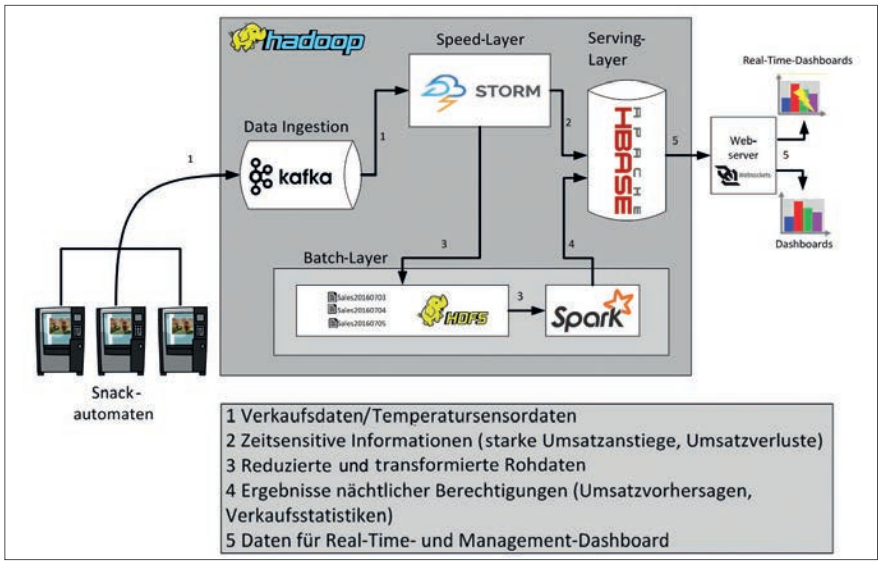

Abb. 5: Die konkrete an der Lambdaarchitektur angelehnte High-Level-Architektur

tigen, wie das ursprünglich in Berlin entwickelte Tool Apache Flink.

Um sämtliche Rohdaten zu speichern, nutzten wir das Hadoop Distributed File System (HDFS), das als Kernbestandteil von Hadoop als Standard für das Speichern großer Mengen von Rohdaten gilt. HDFS kümmert sich selbstständig um die Datenreplikation und sorgt damit für Redundanz und aufgrund der HDFS-Architektur auch für einen hohen Datendurchsatz beim Datenabruf. Auf der anderen Seite hat HDFS allerdings sehr hohe Latenzzeiten beim Datenzugriff. Da die Daten in unserem Fall nur vom nächtlichen Batch-Job benötigt wurden, bei dem der Datendurchsatz eine große Rolle spielt, die Latenzzeit hingegen so gut wie keine, war HDFS hier für uns die beste Wahl.

Schwieriger war die Entscheidung bei der Wahl des Stream-Processing-Frameworks. In die engere Auswahl kamen Apache Storm und Spark Streaming. Apache Storm ist quasi der Dinosaurier unter den Stream-Processing-Frameworks und hat aufgrund seiner komponentenbasierten Herangehensweise ein komplett anderes Handling in der Implementierung als Konkurrenten wie Spark Streaming oder Flink. Der Vorteil von Storm ist, dass es im Gegensatz zu Spark Streaming einen reineren Stream-Ansatz unterstützt und dadurch bessere Kennzahlen im Hinblick auf Reaktionszeiten erzielt. Als Nachteil gilt dagegen der im Vergleich geringere Datendurchsatz von Storm.

Da wir in unserem Anwendungsfall weder einen speziellen Fokus auf den Datendurchsatz setzen mussten, noch die Reaktionsgeschwindigkeit eine herausragende Rolle spielte, waren beide Tools für unseren Anwendungsfall geeignet. Unsere Wahl fiel letztendlich auf Storm, weil Storm im Gegensatz zu anderen Frameworks eine recht geringe Einarbeitungszeit verspricht und in fast allen gängigen Programmiersprachen benutzt werden kann. Auch funktionaler Programmiercode und die MapReduce-Methodik spielen hier keine große Rolle.

Als Datenhaltungssystem für den Serving-Layer wählten wir die NoSQL-Datenbank HBase. Als Wide Column Store ist sie in den meisten Hadoop-Distributionen vorhanden, sie ist gut skalierbar und bietet ein flexibles Datenmodell. Wie bei vielen anderen NoSQL-Datenbanken sind die Schemadefinitionen im Vergleich zu relationalen Datenbanksystemen auch bei HBase unterschiedlich. Die Schemadefinition erfolgt hier nicht anhand fachlicher Definitionen, sondern anhand des Modus, nach dem die Daten später von den Applikationen abgerufen werden. Die Grundsatzfrage lautet hier also nicht „Wie schaffe ich ein Datenbankschema mit sauberer fachlicher Trennung und ohne Redundanzen in vierter Normalform?“ sondern „Wie muss ich die Daten speichern, damit meine Systeme möglichst schnell an die Daten kommen?“. Die resultierenden Datenmodelle sind für Datenbankentwickler mitunter verwunderlich. In unserem Datenmodell werden, wie bei vielen NoSQL-Datenbanken üblich, Joins vermieden, keine Normalformen eingehalten und damit Redundanzen zu Gunsten der Query-Performance in Kauf genommen.

FAZIT

Die dargestellte Architektur zeigt, dass Sie Big-Data-Projekte auf Open-Source-Basis nicht mit einem einzigen Tool realisieren können. Es ist stets ein Zusammenspiel verschiedener Tools nötig. Zwar ist die Investition in eine solche Datenplattform, für die entsprechendes technisches Know-how aufgebaut werden muss, nicht zu unterschätzen, jedoch profitieren Unternehmen von den zahlreichen Vorteilen einer Hadoop-basierten Plattform. Die Vorteile folgen vor allem aus der hohen Skalierbarkeit und der flexiblen Anpassbarkeit auf neue, heute noch nicht definierte Anwendungsfälle. Big-Data-Frameworks sind von Haus aus auf Skalierbarkeit und hohe Geschwindigkeiten ausgelegt. So sparen Sie die regelmäßig steigenden Lizenzkosten für Datenbanken und die Arbeit an Workarounds für Performanceverbesserungen. Hadoop und Co. ersetzen klassische relationale Datenbanken jedoch nicht komplett, sondern sie ergänzen diese – und zwar für die Anwendungsfälle, in denen solche Datenbanken an ihre Grenzen stoßen.

Unsere Erfahrungen aus obigem Prototyp zeigen, dass sich die Implementierung der Logik nicht immer ganz einfach gestaltet. Die Mühen lohnen sich allerdings spätestens, sobald man ein hoch skalierbares System geschaffen hat, das auch für zukünftige Anforderungen gewappnet ist. Und übrigens: Wer sich, um Erfahrungen zu sammeln, nicht gleich ein ganzes On-Premises-Cluster in das Rechenzentrum stellen möchte, für den gibt es jetzt in der Cloud eine Antwort: Amazons AWS bietet mit EMR und Azure mit HDInsight die Möglichkeit, in wenigen Minuten ein Hadoop-Cluster aufzusetzen.

Links & Literatur

[1] Singer, Michael: „The Half-Life of Data and the Role of Analytics“: https://blogs.oracle.com/analyticscloud/the-half-life-of-data-and-the-role-of-analytics

Lukas Berle

beschäftigt sich als Big Data Software Engineering Lead bei OPITZ CONSULTING intensiv mit dem Thema Big Data. Dabei liegt sein Fokus insbesondere auf den Themen skalierbare Anwendungen, Datenspeicherung, In-Memory Stream Processing Engines und deren Integration im Hadoop-Ökosystem.?