Key Value Datenspeicher am Beispiel Redis

Lemmy Marco Tauer

Hochschule Mannheim
Fakultät für Informatik
Paul-Wittsack-Straße 10
68163 Mannheim
lemmy.coldLemonade.tauer@gmail.com

Zusammenfassung. Die Ausarbeitung befasst sich mit noSQL Key-Value Datenspeichern und beschreibt deren funktionale Eigenschaften, die zugrundeliegenden Konzepte und Konsequenzen im Vergleich zum Einsatz relationaler Datenbanken am Beispiel Redis. Anhand der Beispieldatenbank Redis werden Eigenschaften eines Key-Value Datenspeichers aufgezeigt, technische Zusammenhänge sowie Hintergrüne erläutert, deren Verständnis für die eigenständige Anwendung und Problemlösung mit einem Key-Value Datenspeicher erforderlich sind.

Die Verwendung von Key Value Datenspeicher, dem grundlegenden noSQL Datenbanktyp, dient zur effizienten Speicherung anwendungsrelevanter Daten innerhalb einer verteilten Umgebung. Die hohe Nutzerbeteiligung, Interaktivität und Vielgestaltigkeit moderner Plattformen und deren Inhalte, zusätzlich begünstigt durch eine fortwährende Zunahme der verfügbaren Rechenleistung und Speicherkapazität, resultieren in erheblichen Datenlasten webbasierter, wissenschaftlicher und operationaler Anwendungen. Hierzu bedarf es angepasster, effizienter Verfahren zur Datenspeicherung und Verarbeitung in verteilten Anwendungen jenseits der umfangreichen,bekannten relationalen Datenbanken.

1 Themenbeschreibung

1.1 Ziel des Dokuments

Es geht darum, wann und auf welche Weise der Einsatz von Key-Value Datenspeichern in einem verteilten System sinnvoll ist und die Eigenschaften der Infrastruktur und Methodik so ergänzt, dass ein effektiver Nutzen für den Anwender, Betreiber und Entwickler entsprechender Webanwendungen, Dienste und Systeme resultiert. Daher ist eine grundlegende Kenntnis über die Funktionsweise eines Key-Value Datenspeichers unerlässlich um dessen Eignung für eine bestimmte Problemdomäne erkennen zu können. Diese Kenntnise zu vermitteln und einen Ausgangspunkt für vertiefende Recherchen zu bilden ist der Zweck dieser Ausarbeitung (Bonmassar, 2011).

1.2 Problembeschreibung

Spezifische Anforderungen an verteilte Datenspeicherungssysteme kollidieren mit der formalen Arbeitsweise relationaler Datenbanksysteme. Diese sind auf die Abbildung (transitiver) verschachtelter Abhängigkeiten mit Hilfe der zeilenorientierten Datensätze in Tabellen ausgelegt. Dabei werden dem Entwickler strukturelle Vorgaben auferlegt, die Einfluss auf den Entwurf sowie funktionale Eigenschaften der angestrebten Lösung nehmen. Die Effizienz einer relationalen Datenbank nimmt mit zunehmender Datenmenge unverhältnismäßig ab, nachträgliche Veränderungen (aposteriori) der Datenstruktur sind sehr ineffizient Cattell (2011). Die strikten Konsistenzbestimmungen führen zur Beeinträchtigung der Leistungsfähigkeit und Skalierbarkeit bei verteilten Zugriffen. Es ist abzuwägen, unter welchen Bedingungen zu Gunsten der Leistungsfähigkeit und Partitionierbarkeit des Speichersystems Konsistenzrestriktionen aufgeweicht werden können.

1.3 Vorgehensweise

Unter Berücksichtigung von Analogien und Unterschieden zwischen relationalen Datenbanken und nicht-relationalen Datenbanken ist eine Bemessung der Eignung einer bestimmten Lösung im allgemeinen erstrebenswert. Bei der Speicherung und Verarbeitung großer, sehr vielgestaltiger Datenvolumen, sogenannter Big Data, führt das relationale Schema zu Effizienzeinbusen bei der Entwicklung und im Betrieb von Anwendungen. Genauso verhält es sich mit stark dynamischen kleinen Datenmengen, die durch leichtgewichtige Ansätze an Leistung und Flexibilität gegenüber umfangreichen Datenbanken gewinnen. Eine genau Kenntnis der Motive hinter relationalen und nicht-relationalen Datenbanken ermöglicht es für Anwendungsbereiche die adäquate Vorgehensweise und Technologie zielgerichtet auszuwählen Tiwari (2011).

2 Technologische Begriffe

In diesem Abschnitt werden notwendige Begriffe, die für das Verständnis der Betrachtungen dieser Seminararbeit notwening sind, beschrieben. Für weiterführende, vertiefende Erläuterungen sei auf das Literaturverzeichnis verwiesen.

2.1 Big Data

Big Data bezeichnet die stetig wachsenden Datenmengen moderner Anwendungen mit geringer vordefinierter Strukur, bezeichnet als schwach schematisch oder schemalose Daten. Big Data steht für die immensen Datenmengen, die in täglichen Arbeits- und Wirtschaftsprozessen erzeugt werden. Durch die umfangreiche Vernetzung von Kommunikationsplattformen, Sensoren und Datenverarbei-

tungssystemen erzeugte Daten sind Big Data. Analyse und Verarbeitung von Big Data unterliegen erhöhten Anforderungen an Datendurchsatz und Datenspeicherkapazitäten. Sie müssen effizient abgelegt und verwaltet werden, was eine Verteilung und parallele Verarbeitung impliziert. Hier können relationale Datenbanken keine flexible Lösung hinsichtlich horizontaler Skalierung darstellen. Häufig ist Big Data das Ergebnis von Echtzeitprozessen die Daten erzeugen, weiterverarbeiten und zusammenstellen. Dies geschieht neuerdings maßgeblich im Bereich der Webanwendungen mit hoher Nutzerinteraktion und vielfältigen Inhalten und bei der Analyse von Geschäftsabläufen und Prozessstrukturen. Was als Big Data bezeichnet wird hängt zudem maßgeblich vom aktuellen Stand der Technik ab. Derzeit beginnt man ab ca. 50 Terrabyte von Big Data zu sprechen. Diese Daten liegen nicht in der strukturellen Definiertheit vor, wie diese von relationalen Modellen effizient abgebildet werden (Zikopoulos et al., 2012).

2.2 noSQL

NoSQL ist ein Sammelbegriff moderner (und vergessener) nicht-relationaler Technologien zur Speicherung und Manipulation großer Datenmengen, wie sie u.a. in Webanwendungen, Simulationen und unterschiedlichen Analysen auftreten. Ihr Erscheinen ist auf die Herausforderungen im Umgang mit Big Data zurückzuführen. Geprägt von Entwicklungen bei Google, Amazon und Yahoo (2005- heute) bezeichnet noSQL das Bestreben überall dort neue Verfahren anzuwenden, wo klassische relationale Datenmodelle zu restriktiv und unflexibel sind. NoSQL Datenbanken dienen vorrangig der verteilten Datenspeicherung und sind Teil paralleler Datenverarbeitungssysteme. (Tiwari, 2011)

2.3 CAP Theorem

Von Konsistenz, Verfügbarkeit und Partionierbarkeit (Consistency, Availability, Partitioning Capability) können in einem verteilten Datenspeicher maximal 2 Eigenschaften gleichzeitig erfüllt werden, während die 3. Eigenschaft beeinträchtigt bzw. deren Einhaltung ausgeschlossen ist. Es existieren umfangreiche Abhandlungen über dieses Thema, auf die hier für vertiefende Darstellungen verwiesen wird (Browne, 2009). Verfügbarkeit und Partionierbarkeit sind die Schwerpunkte von noSQL gegenüber Konsistenz und Verfügbarkeit als Schwerpunkt der relationalen Datenbanken.

2.4 ACID und BASE Semantik

Atomarität, Konsistenz, Integrität und Dauerhaftigkeit (Atomicity, Consistency, Integrity, Durability) sind die 4 Grundmerkmale von Datenoperationen, die in relationalen Datenbanken eine lange Tradition besitzen. Im Bereich verteilter Datenspeicher wird das ACID Kriterium aufgeweicht, weil der Aufwand zur

globalen Synchronisation verteilter Daten zu hoch ist, um den Leistungsanforderungen gerecht zu werden. Dies beruht auf der Aussage des CAP-Theorems (Theo Haerder, 1999). Dadurch ergibt sich im Bereich der noSQL Datenbanken die sogenannte BASE Semantik (Basically Available Soft-State Eventual-Consistent) auf datenlogischer Ebene, die permanente Verfügbarkeit durch verzögerte globale Konsistenz ermöglicht, zugunsten einer verbesserten Partitionierbarkeit (Pritchett, 2008).

2.5 CRUD

Erstellen, Lesen, Verändern und Löschen (Create, Read, Update und Delete) sind die atomaren Operationen auf Daten innerhalb jeder Datenbank. Allen noSQL Datenbanken ist gemeinsam, dass Erstellen und Lesen die wichtigen Befehle bilden, die Verändern und Löschen Operationen vorgezogen werden. Manche Datenbanken besitzen nur Erstellen und Lesen Befehle für ihre Funktionalität (Tiwari, 2011).

2.6 Verteilte Konzepte

Die Konzepte zur Verteilung von Programmen und Daten sind Gegenstand anhaltender Entwicklungen seit 1970. 1980 etablierte sich das Client-Server Paradigma, bei dem ein Rechner funktional die zentrale Rolle des Servers einnimmt, der Clientverbindungen akzeptiert und Dienste sowie Daten bereitstellt. Der TCP/IP Protokoll Standard ermöglichte einheitliche Verbindungen zwischen den Systemen. Das Peer-to-Peer Paradigma beschreibt die funktionale Vereinigung von Server und Client Eigenschaften in allen Systemen der verteilten Umgebung. Moderne WebServices nutzen Standards zur Definition der Daten, Nachrichtenformaten und Zugriffsmustern, um in einer verteilten Infrastruktur Softwaredienste umzusetzen (Thomas Diekmann, 2003).

2.7 Verteilte Datenspeicher

Bei der Verarbeitung und Speicherung verteilter Daten sind Konsistenz, Verfügbarkeit und die Aufteilbarkeit der Daten innerhalb des Systems besonders grundlegende Merkmale zur Beschreibung der Eigenschaften und Leistungsfähigkeit einer Lösung zur verteilten Datenspeicherung. Ziel von noSQL Datenspeichern ist es die strikte Einhaltung der globalen Konsistenz relationaler Systeme abzuschwächen, um eine effizientere, schnellere Aufteilung der Daten bei ständiger Verfügbarkeit zu ermöglichen. Dabei sind die Daten letzlich logisch Konsistent, ohne davor jedoch zu Gewährleisten, dass die Daten global gleichzeitig aktualisiert werden. Moderne Webanwendungen wie Soziale Netzwerke und Multimedia Portale, stellen vielen Millionen Nutzern weltweit Anwendungsdaten zur Verfügung, um diese in Echtzeit einzusehen sowie zu verändern. Dabei ist es zumutbar, dass Veränderungen der Daten nicht überall strikt gleichzeitig sichtbar sind.

Die Menge gespeicherter und berechneter Daten nimmt stetig zu, weil u.a. die Kapazität verfügbarer Speichermedien beständig ansteigt. Die Zugriffsgeschwindigkeiten und Datentransferraten halten dieser Entwicklung nicht Stand. Allein zur rechnerischen Bewältigung massiver Datenmengen, ist eine Verteilung der Daten auf viele, verteilte Speichermedien notwendig, um eine angemessene Leistung des Datenspeichers für verteilte Zugriffe zu gewährleisten (Mearian, 2010). Außer Geschwindigkeitsüberlegungen führt auch eine Betrachtung der Ausfall- und Datensicherheit zur Notwenigkeit einer Verteilung der Daten unter den vorhandenen Speichersystemen, um die Toleranz des Systems gegenüber Fehlern zu erhöhen. Die Verteilung führt zu komplexen Synchronisationsproblemen und Verfahren zur Gewährleistung eines definierbaren Zustands der gespeicherten Daten. Es existieren verschiedene Stufen zur Beschreibung des Umgangs mit redundanten Daten in einem verteilten Datenspeicher. Hierbei handelt es sich um Restriktion auf notwendige Strategien zur Erreichung eines definierten Grades an die Konsistenz der simultan bearbeiteten Daten. Besonders Kentniss der verwendeten Cachingverfahren und den Definitionen zur Sicherheit bzw. Gültigkeit der Daten stellt ein großes Potential zur Verbesserung der Leistung eines verteilten Systems dar. Insofern nicht für die nähere Betrachtung der technischen Details notwendig, wird auf bestimmte Merkmale zur Veranschaulichung der Gegebenheiten lediglich schematisch eingegangen (Class, 1997).

2.8 relationale Datenbanken

Relationale Datenbanken sind umfangreiche Werkzeuge, sie bieten effiziente Anfragen und kompakte Referenzierung. Sie setzen aber vorraus, dass Daten entsprechend normalisiert werden, deren genau Struktur bereits bekannt ist und jedes Element eindeutig identifizierbar ist. Dies geschieht über ein zeilenorientierte Anordnung der Daten in Tabellen, deren Indizierung und Referenzierung über Primär/Fremdschlüssel erfolgt, die Verweise in Beziehung (Relation) stehender Tabellen bilden. Die leistungsfähige Abfragesprache SQL mit Joins und verschachtelten Abfragen, verfügt über DDL + DML zur Verarbeitung und Manipulationen sehr komplexer, relationaler Datensätze. In der jüngsten Zeit hat sich die Betrachtung vom Allround Werkzeug der relationalen Modellierung gewandelt, im Bereich der enorm anwachsenden nur schwach apriori zu schematisierender Daten, die den Großteil der Daten in Webandwendungen wie Facebook, Googler oder Twitter ausmachen, haben sich nicht relationale Ansätze als leistungsfähig und effizient erwiesen. Die Notwendigkeit einer eindeutigen Indizierung bleibt in der Regel auch bei noSQL Datenbanken bestehen, um effiziente und integere Operationen auf Daten zu gewährleisten. Fest definierte strukturelle Vorgaben gelten vielmals für Szenarien im Bereich datenintensiver, paralleler Webanwendungen nicht. Daher befasst sich ein weites Feld an Entwicklern mit Untersuchungen über verteilte Algorithmen, verteilte Entwurfsmuster und verteilte Datenverarbeitung, sowie Verfahren zur Verbesserung vitaler Laufzeitparameter, der Effizienz der Verarbeitung, der Parallelisierbarkeit der Prozesse und der Skalierbarkeit. Dem zugrunde liegt das Bestreben bereits zu Mainf-

rame Zeiten bestehende Verfahren zu verbessern, bekannte und grundlegende Modelle und Konzepte neu zu bewerten und die Effizienz in der Anwendung und Entwicklung sowie Administration von verteilten Softwaresystemen und deren profundem Datenmanagement zu verbessern (Richling, 2011).

2.9 nicht relationale Datenbanken

Der Bereich der nicht relationalen Datenbanken ist ein aktives Entwicklungsfeld. So gibt es unterschiedliche Ansätze, die unterschiedliche Besonderheiten aufweisen. Im Kontext dieser Ausarbeitung seien die 3 prominentesten Kategorien kurz angesprochen. Sogenannte Spaltendatenbanken (Column-Oriented Databases) (wie HBase, Hypertable, Cloudata, ...), Key Value Stores (Memcached, Membase, Redis, Cassandra, Kyoto Cabinet, ...) und Document Stores (MongoDB und CouchDB). Key-Value Datenspeichern sind das Zentrum der Betrachtung. Spaltendatenbanken stellen in ihrem Aufbau eine Erweiterung der Key-Value Datenspeicher dar, indem Sie technisch gesehen, Key-Value Felder assoziativ verknüpfen. Daher wird dieser Zusammenhang in die Überlegung einbezogen. Ursprung und Funktionsweise dieser 2 Kategorien von noSQL Datenbanken werden kurz im folgenden Abschnitt geschildert (Tiwari, 2011).

2.10 Key Value Datenspeicher

In der Softwareentwicklung sind *Key-Value Container* spätestens seit der Einführung der STL von C++ oder entsprechender Containerklassen anderer Programmiersprachen sehr gebräuchlich. Bekannte Key-Value Container sind Hashmaps und assoziative Arrays (Schildt, 2002). Sie besitzen günstige Laufzeiteigenschaften als Datenstrukturen, eine konstante Laufzeit von $O(1)$ zum Zugriff auf und $O(log(n))$ zum Einfügen und Löschen eines Elements. Bei der Speicherung von Big Data, ergeben sich vorteilhafte Laufzeitcharakteristika bei der Verwendung von Key-Value Datenspeichern(Yadav, 2007). Exemplarische Vertreter der Key-Value Datenspeicher sind BerkleyDB, eine sehr einfache und schnelle Implementierung, auf der Basis von Bytearrays, Memcached, eine sehr verbreitete Lösung, u.a. zum Puffern von SQL Abfragen, basierend auf Hashmaps, die ausschließlich im Speicher gehalten werden, d.h. es existiert keine Persistenz in Memcached und Redis, das in dieser Ausarbeitung näher Betrachtet wird. Es handelt sich um einen leistungsfähigen Key-Value Datenspeicher, der Memcached in seinem Aufbau ähnelt und zusätzliche Funktionen und Datentypen bietet. Daten liegen im Arbeitsspeicher und werden asynchron auf Platte persistiert. Redis unterstützt Strings, Lists, Sets, Sorted Sets und Hashmaps. Hier beginnen die Grenzen zwischen Spaltendatenbanken und Key-Value Datenspeichern zu verschwimmen. Der technische Zusammenhang wird im folgenden Kapitel noch genauer betrachtet. Redis ist einer der schnellsten Key-Value Datenspeicher und als single-threaded Server für Linux entwickelt. Mittlerweile ist es so populär, dass Microsoft Open Technology Redis nach Windows portiert hat (Frommel,

2012). Daneben spielt auch Amazons Dynamo eine wichtige Rolle, das populäre Open-Source Abkömmlinge hervorgebracht hat und den Begriff der *Schließlichen Konsistenz* (Eventual Consistency) prägt. Hierbei werden im Zusammenhang des CAP Theorems Zugeständnisse an die strikte Konsistenz zugunsten erhöhter Laufzeitgeschwindigkeit und horizontaler Skalierbarkeit gemacht Tiwari (2011).

2.11 Spaltendatenbanken

BigTable ist die *Spaltendatenbank*(Column-Oriented Database) von Google. Sie ist mit der Veröffentlichung der zugehörigen Google Paper (Chang et al., 2006) zur Inspiration der noSQL Bewegung geworden und dient als Vorlage für Open Source Projekte wie HBase, Hypertable und Cloudata. Hadoop ist ein Open Source Projekt angelehnt an die Google Plattform und bietet eine verteilte Infrastruktur mit verteiltem Dateisystem und Master-Worker Schema. HBase ist Teil des Hadoop Projekts und verfügbt mit ZooKeeper über ein eigenes Verwaltungssystem mit GUI Tiwari (2011). Intern arbeitet HBase mit verschachtelten Hashmaps, ist also einer Erweiterung des Key-Value Datenspeicher Funktionsprinzips (Chang et al., 2006).

3 Key Value Datenspeicher

Der folgende Abschnitt handelt von den Eigenschaften eines Key Value Datenspeichers. Key Value Datenspeicher lassen sich in die Kategorie der noSQL Datenbanken einordnen. Datensätze werden in Key Value Paaren gespeichert, bilden eine einfache Struktur. Key Value Datenspeicher bilden Eigenschaften heraus, die in relationalen Systemen aufgrund der expliziten Abbildung von Abhängigkeiten (Zusammenhängen) in der Datenstruktur und Anforderungen an Konsistenz und Flexibilität bei der Datenabfrage und Manipulation nicht erreichbar sind. Die einfache Struktur ermöglicht schnelle Zugriffe bei Lese- und Schreiboperationen gegenüber einer komplexen, spezifischen relationalen Datenstruktur (Heise, 2010).

3.1 Beispieldatenbank: Redis

Redis, entwickelt von Salvatore Sanfilippo und erstmals 2009 als Open-Source veröffentlicht, ist ein schneller Key-Value Datenspeicher, der vorrangig zur Speicherung veränderlicher anwendungsrelevanter Daten mit verteiltem Zugriff verwendet wird. Seit 2010 wird Redis von VMWare weiterentwickelt, wo die Core-Entwickler Sanfilippo und Noordhuis nun hauptberuflich an dem Open-Source Projekt arbeiten (Sanfilippo, 2010b).

3.2 Grundmerkmale

Bei Redis handelt es sich um einen verteilten In-Memory Datenspeicher, der für hohe Lese- und Schreibgeschwindigkeit entwickelt wurde. Geschwindigkeit, Skalierbarkeit und Einfachheit sind die generellen Hauptmerkmale des Redis Datenspeichers. Schreibzugriffe sind in Redis schneller als Lesezugriffe, wodurch er sich für Szenarien mit entsprechenden Zugriffsmustern zusätzlich eignet Die einfache Struktur der Key-Value Daten lässt Redis gut horizontal skalieren.

3.3 Laufzeitbibliotheken und Clients

Laufzeitbibliotheken zur Entwicklung mit Redis sowie Clients existieren für alle verbreiteteten Programmiersprachen, wie C,C++,C#, Erlang, Go, Java, LUA, Perl, Python, PHP u.a. Eine aktualisiert, offizielle Zusammenstellung der verfügbaren Clients findet sich unter (Redis, 2012b).

3.4 Plattform und Lizenz

Redis ist in ANSI-C entwickelt und steht unter der Berkley Software Distribution (BSD) Lizenz mit geöffnetem Quellcode (Open-Source) und wurde erstmals 2009 veröffentlicht. Die offizielle Version steht als Download für Linux zur Verfügung, die verfügbaren Windows 32 und 64Bit Versionen werden als inoffiziell geführt, jedoch von offizieller Seite unterstützt. Die derzeit aktuellste Version ist der RC5 der Version 2.6 sowie die stabile Version 2.4.15 vom 21.06.2012 (Redis, 2012f). Der Redis-Cluster, ein Framework zur Verwendung von Redis als fehlertolerantes, verteiltes Speichersystem steht hinter den Entwicklungen von Redis Version 2.6 zurück, eine erste Veröffentlichung des Redis-Cluster ist für 2012 geplant.

3.5 Funktionsweise

Redis speichert Daten in Key-Value Datensätzen. Die Daten bestehen aus Byte-Strings, die beliebige Binärdaten speichern können. Neben (Byte-)Strings existieren die Datenstrukturen Hashes/Assoziatives Array, Listen, ungeordnete und geordnete Mengen. Zugriffe mit Schlüssel erfolgen in konstanter Zeit (Seguin, 2010). Diese Funktionalität ermöglicht eine effiziente Speicherung von objektorientierten und prozeduralen Anwendungsdaten (Jansen, 2011). Zum Zugriff auf die gespeicherten Daten, stehen atomare Operationen wie SET, CREATE, READ und DELETE zur Verfügung. Die Atomarität aller serverseitigen Funktionen ermöglicht den verteilten Zugriff auf den Datenspeicher durch Clientsysteme mit selbst implementierter Synchronisation (Willison, 2010). Es können dabei asynchrone und synchrone Zugriffsfunktionen verwendet werden.

3.6 Serverseitige Operationen

Zu jedem Datentyp existieren viele serverseitige Funktionen, die Operationen auf die gespeicherten Daten zur Verfügung stellen. Eine vollständige Liste aller Redis Befehle findet sich in der Redis Dokumentation. Keine der Operationen besitzt die Laufzeiteigenschaft $O(n^2)$ oder $O(C * n)$. Eine genaue Auflistung des Laufzeitverhaltens der einzelnen Operationen in Abhängigkeit bestimmter Parameter sind in der Redis Dokumentation enthalten. Alle serverseitigen Operationen sind atomar und lassen sich zusätzlich mit Transaktionen zusammenfassen und als atomare Einheit auf dem Server ausführen, so dass kein anderer Client während der Sequenz Zugriff auf die Daten erhält. Die Reihenfolge der Operationen wird sichergestellt und die Transaktion ist entweder ganz erfolgreich oder schlägt fehl (Redis, 2012d).

3.7 Hashes

Hashes sind Assoziative Arrays, eine Datenstruktur, die einem Schlüssel einen entsprechenden Inhalt zuordnet. Als Schlüssel werden aus dem Inhalt generierte Hashwerte verwendet, die einen Zugriff auf Datenelemente in konstanter$O(1)$ Zeit ermöglichen (Redis, 2012e). Hingegen dienen aus den Schlüsseln generierte Hashwerte zur Verteilung der Daten auf verschiedene Redis-Server. Durch den Hashwert lässt sich Einfluss auf diese Verteilung nehmen, um Daten entsprechend ihres Inhalts und ihrer Verwendung anzuordnen (Zawodny, 2011). Das Projekt Redis-Cluster beschäftigt sich mit der automatisierten Verteilung, Replikation und Integration von Redis in einem Cluster (Redis, 2012c).

3.8 Persistenz

Die Daten werden zyklisch in RDB Dateien als Schnappschüsse gespeichert und Veränderungen in einem Journal festgehalten, um Systemausfälle ohne Datenverlust zu gewährleisten. Bei steigendem Datenvolumen führt das zu hohen I/O Lasten des Speichersubsystems, den Festplatten. Die persistierten Daten können dabei komprimiert werden, um das Datenvolumen der Daten zu reduzieren, zusätzlich lassen sich die Zyklen zur Speicherung anpassen, um Leistung zu Lasten der potentiellen Datensicherheit zu gewinnen (Zawodny, 2011).Redis unterstützt desweiteren das Persistieren bei jeder Veränderung durch eine AOL Datei an die nur Daten angefügt werden können (Append-Only). Diese Datei ist inkrementell aufgebaut und lässt den Server nach einem Ausfall die Daten rekonstruieren. AOL Dateien sind größer als der Schnappschuss der RDB Datei (Redis, 2012h).

3.9 Replikation

Redis ermöglicht die Replikation eines Redis-Servers auf beliebig viele weitere Server, die keine dauerhaften Schreibzugriffe zulassen, jedoch Anfragen verarbei-

ten können. Die Replikate werden bei jeder Veränderung aktualisiert. Im Falle eines Systemausfalls muss die Konfiguration per Hand angepasst werden, um den ausgefallenen Server zu ersetzen, bzw. mit eigenen Skripten automatisiert werden. Ein Replikatsserver kann anstelle des Redis-Servers die Persistierung der Daten durchführen, um die Datensicherheit zu erhöhen und gleichzeitig die Geschwindigkeit des Redis-Servers zu gewährleisten(Seguin, 2010).

3.10 Beobachter Muster

Redis unterstützt das Beobachter Muster auch bekannt als Publish-Subscribe Verfahren, wodurch Clients sich benachrichtigen lassen können, wenn sich einem Schlüssel oder Schlüsselbereich zugeordnete Werte verändert haben (Sanfilippo, 2010a). Clients können Daten auf Kanäle senden und Eingaben verschiedener Kanäle verarbeiten (Seguin, 2010).

3.11 Leistungsmessung und Überwachung

Redis bietet ein integriertes Überwachungswerkzeug, den Redis-Monitor, der alle ausgeführten Befehle des Redis Servers ausgibt und hilft die genauen Abläufe im Falle von Fehlern oder zum Zweck der Optimierung einzusehen. Da der Monitor alle Anfragen an den Server ausliest, erhöht sich die benötigte Transferleistung des Servers und der Datenübertragung (Redis, 2012g). Die Leistungsfähigkeit eines Redis Servers als Instanz auf einem bestimmten Hardwaresystem lässt sich mit dem Redis-Benchmark ermitteln. Das Werkzeug dient zur Bemessung der theoretischen Leistungsfähigkeit der Instanz und als Vergleichsgröße zwischen unterschiedlichen Hardwaresystemen, nicht jedoch zum Vergleich von Redis mit anderen Datenspeichersystemen. Mit Hilfe des Redis-Benchmarks lässt sich feststellen, ob eine Konfiguration erwartungsgemäß ausreichend leistungsfähig ist (Redis, 2012a).

3.12 Vor- und Nachteile

Redis bietet durch seine integrierten Datentypen wie Listen, Mengen und Hashes für Programmierer eine vertraute Umgebung und erfordert geringen Aufwand zur Aneignung der nötigen Kenntnisse zum Einsatz von Redis. Die Verfügbarkeit für alle gängigen Programmiersprachen ermöglicht eine effiziente Integration in vertraute Arbeitsweisen und vorhande Softwaresysteme. Da Redis alle Arbeitsdaten im Speicher hält, bietet Redis schnelle Lese- und Schreibperformance für verteilte Zugriffe. Redis unterstützt ein Transaktionsschema, ‚dass deterministisches Verhalten für verteilte Zugriffe ermöglicht. Komplexe Datensätze und Abfragen sind nicht die Stärken von Redis, hierfür eignen sich relationale Datenbanken oder komplexere noSQL Varianten. Laufzeitdaten für verteilte Anwendungen speichert Redis hingegen sehr effizient und schnell. Es lässt sich als intuitives

Speichersystem in einer Vielzahl von Anwendungen verwenden. Der Speicherbedarf kann als kritisch angesehen werden, da Redis alle Daten im Speicher hält. Eine Verteilung der Daten auf verschiedene Server ist auf Anwendungsebene umsetzbar.

3.13 Einsatzbereiche

Redis eignet sich durch seine Geschwindigkeit für viele Einsatzbereiche, die keine komplexen Datenstrukturen erfordern. Redis kann als schneller Publish/Subscribe Server oder als System zum Datenaustausch zwischen Anwendungen verwendet werden. Eine umfangreiche Liste interessanter Anwendungsbereiche findet sich bei (Meyer, 2010).

3.14 Perspektive

Automatisierte Fehlerbewältigung, gleichmäßige Datenverteilung in einem Verbund aus Servern und horizontale Skalierung sind Bereiche, in denen Redis über keine entsprechende eigene verteilte Softwareinfrastruktur verfügt. Das Redis-Cluster Projekt befasst sich mit dieser Problematik. Bis zur anstehenden Veröffentlichung lassen sich Replikation und Partitionierung nur durch eigene Implementierungen fehlertolerant in einem Verbund aus Redis Servern vereinen (Redis, 2012c).

4 Schlusswort

Es gibt schnelle Datenspeicher für den Einsatz in verteilten Anwendungen. Einer dieser Datenspeicher ist der hier vorgestellte Key-Value Datenspeicher Redis. Es gibt keine generellen Alleskönner Datenspeicher, die für jeden Einsatzbereich ideal geeignet sind. Moderne Lösungen erfordern ein breiteres Verständnis der tatsächlichen Anforderungen und oftmals eine Reduktion oder Vereinfachung, um effizient gelöst zu werden. Redis bietet intuitive Datenstrukturen, ist sehr schnell, wird stetig weiterentwickelt und kann mit vielen Programmiersprachen genutzt werden. Es ist nicht mehr sinnvoll, sein Problem in ein generelles Schema zu stecken, sondern vielmehr aus einer Vielzahl verfügbarer Modelle ein für den Problembereich passendes zu finden. Das erfordert Kenntnis über vorhandene Technologien und die Bereitschaft zu neuen Denkweisen. Key-Value Datenspeicher sind eine lohnenswerte Alternative.

Literaturverzeichnis

Luca Bonmassar. [anti]patterns with nosql, 03 2011. URL http://www.slideshare.net/bonmassar/patterns-antipatterns-with-nosql.

Julian Browne. Brewers cap theorem, 01 2009. URL http://www.julianbrowne.com/article/viewer/brewers-cap-theorem/.

Rick Cattell. Scalable sql and nosql data stores. http://cattell.net/datastores/Datastores.pdf, December 2011.

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. Technical report, Google Inc., 2006. URL http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/de//archive/bigtable-osdi06.pdf.

Christina Class. Synchronization issues in distributed applications definitions , problems and quality of synchronization, 12 1997. URL http://e-collection.library.ethz.ch/eserv/eth:24870/eth-24870-01.pdf. Abstract.

Oliver Frommel. Microsoft veroeffentlicht redis fuer windows, Mai 2012. URL http://www.admin-magazin.de/News/Microsoft-veroeffentlicht-Redis-fuer-Windows.

Heise. Key-value datenbank redis, 05 2010. URL http://www.heise.de/open/artikel/NoSQL-im-Ueberblick-1012483.html?artikelseite=2.

Rudolf Jansen. Nosql: Key-value-datenbank redis im ueberblick, April 2011. URL http://www.heise.de/developer/artikel/NoSQL-Key-Value-Datenbank-Redis-im-Ueberblick-1233843.html.

Lucas Mearian. Data growth remains it's biggest challenge, gartner says, 11 2010. URL http://www.computerworld.com/s/article/9194283/Data_growth_remains_IT_s_biggest_challenge_Gartner_says.

Mathias Meyer. A collection of redis use cases, 02 2010. URL http://www.paperplanes.de/2010/2/16/a_collection_of_redis_use_cases.html.

Dan Pritchett. Base: An acid alternative, 05 2008. URL http://queue.acm.org/detail.cfm?id=1394128.

Redis. How fast is redis, 06 2012a. URL http://redis.io/topics/benchmarks.

Redis. Redis clients, 06 2012b. URL http://redis.io/clients/.

Redis. Redis cluster specification, 06 2012c. URL http://redis.io/topics/cluster-spec/.

Redis. Redis command reference, 06 2012d. URL http://redis.io/commands.

Redis. Redis data-types, 06 2012e. URL http://redis.io/topics/data-types.

Redis. Redis download, 06 2012f. URL http://redis.io/download/.

Redis. Redis monitor, 06 2012g. URL http://redis.io/commands/monitor.

Redis. Redis persistence, 06 2012h. URL http://redis.io/topics/persistence.

Tobias Richling. Sql oder nosql, 12 2011. URL http://it-republik.de/dotnet/artikel/SQL-oder-NoSQL-4255.html.

Salvatore Sanfilippo. Redis weekly update no.3 - pub/sub and more, 03 2010a. URL http://antirez.com/post/redis-weekly-update-3-publish-submit.html.

Salvatore Sanfilippo. Vmware: The new redis home, 03 2010b. URL http://antirez.com/post/vmware-the-new-redis-home.html.

Herbert Schildt. *The Complete C++ Reference*. Mcgraw-Hill Professional, 4 edition, Dezember 2002.

Karl Seguin. *The Little Redis Book*. Karl Seguin, 2010. URL http://openmymind.net/redis.pdf.

Erhard Rahm Theo Haerder. Datenbanksysteme - konzepte und techniken der implementierung, 01 1999. URL http://dbs.uni-leipzig.de/buecher/DBSI-Buch/inhalt.html.

Svenja Hagenhoff Thomas Diekmann. Verteilte systeme: State of the art, 01 2003. URL http://www2.as.wiwi.uni-goettingen.de/getfile?DateiID=396.

Shashank Tiwari. *Professional NoSQL*. John Wiley & Sons, Inc., 2011.

Simon Willison. Redis tutorial at nosql europe conference, April 2010. URL http://simonwillison.net/static/2010/redis-tutorial/.

Rajinder Yadav. Stl guide - a tutorial and reference, September 2007. URL http://devmentor.org/references/stl/stl.php.

Jeremy Zawodny. Redis sharding at craigslist, 02 2011. URL http://blog.zawodny.com/2011/02/26/redis-sharding-at-craigslist/.

Paul C. Zikopoulos, Chris Eaton, Dirk deRoos, Thomas Deutsch, and George Lapis. *Understanding Big Data - Analytics For Enterprise Class Hadoop And Streaming Data*. McGraw-Hill, 2012. URL http://www-01.ibm.com/software/data/bigdata/.