Redis

Stefan Dorenkamp
Universität Siegen
stefan.dorenkamp@student.uni-siegen.de

ABSTRACT

Die bis heute weitverbreiteten relationalen Datenbanken, lassen keine effiziente Verwaltung von sehr großen, verketteten Datenbeständen zu. Aus diesem Grund haben sich viele nicht-relationale Datenbanken etabliert, die unter dem Begriff *NoSQL-Datenbanken* zusammengefasst werden. Allerdings setzen diese zum Teil auf sehr unterschiedliche Technologien, weshalb in dieser Ausarbeitung ein Einblick in die Datenverwaltung mit Redis gegeben wird. Neben der grundlegenden Funktionsweise, sowie deren Vor- und Nachteile, werden denkbare Einsatzszenarien dargelegt. Abschließend soll dadurch die Entscheidung möglich sein, ob Redis für einen gegebenen Anwendungsfall, zur Datenverwaltung geeignet und empfehlenswert ist.

1. EINLEITUNG

Obwohl NoSQL-Datenbanken grundsätzlich die Gemeinsamkeit haben, keine Tabellenschemata und Relationen zu verwenden, arbeitet jede für sich mit unterschiedlichen Techniken und ist aus einer anderen Motivation heraus entstanden. Dies ist der Grund dafür, warum sich mit der Zeit zahlreiche, verschiedene Datenbanklösungen mit einem NoSQL-Hintergrund gebildet haben.

Um gerade für moderne Anwendungen, im Bereich von Big Data oder dem Web 2.0, eine passende Datenbank auszuwählen, sollten grundlegende Kriterien schon bei der Entwicklung feststehen. Anhand dieser lässt sich die Zahl der möglichen Kandidaten schnell eingrenzen, wodurch vor allem vermieden wird eine gänzlich falsche Entscheidung zu treffen. Letzteres würde einen hohen Mehraufwand bedeuten, da man unter Umständen die komplette Struktur der Daten verändern muss, um diese an eine neue Datenbank anzubinden.

Warum Redis entwickelt wurde, welcher Hintergrund dabei wichtig war und wie die aktuelle Entwicklung aussieht, wird im zweiten Kapitel genauer betrachtet. Darauf folgt bereits die eigentliche Funktionsweise im dritten Kapitel, aus der auch die Vor- und Nachteile von Redis abgeleitet werden. Dort wird jedoch nicht nur die Verarbeitung der Daten betrachtet, sondern ebenso der Fokus auf die dauerhafte Speicherung und den Einsatz als verteiltes System gelegt. Zudem werden einige Besonderheiten erläutert, die bei der Datenverwaltung mit Redis zu empfehlen sind.

Darauf aufbauend wird der praktische Einsatz von Redis erläutert, angefangen bei der Installation und Einrichtung einer Datenbank, über die Einbindung in eigene Softwareprojekte, bis hin zu verschiedenen Verwendungsmöglichkeiten. Im vierten Kapitel werden nicht nur hilfreiche Punkte für die praktische Nutzung von Redis genannt, denn darüber hinaus erfolgt ein kurzer Blick auf die Performance.

2. ENTWICKLUNGSGESCHICHTE

2.1 Hintergrund

An der Entwicklung von Redis ist, seit der ersten Version im Jahr 2009, maßgeblich der Begründer Salvatore Sanfilippo beteiligt. Dieser arbeitete zuvor, gemeinsam mit Fabio Pitrola, an LLOOGG[1] und benötigte eine Alternative zu MySQL, um größere Datenmengen in kurzer Zeit verarbeiten zu können [1]. Letztlich begann er selbst, mit der Arbeit an einer neuen Datenbank und wurde damit zum Erfinder von Redis. Der Name steht hierbei für *Remote Dicitionary Server* und die Programmierung erfolgte komplett in C[2].

2.2 Wandlung zu Open Source

Nachdem Sanfilippo die grundlegende Arbeit an der Server Software beendet hatte, entschloss er sich, diese als Open Source Projekt öffentlich zur Verfügung zu stellen. Seine Motivation dabei war letztlich nur, dass ihm der Gedanke gefiel seine Arbeit mit anderen Interessierten zu teilen[3].

Durch die Veröffentlichung des Quellcodes, erreichte er verschiedene Vorteile. Zum einen wurde Redis von anderen Anwendern in verschiedenen Einsatzszenarien getestet, wodurch eine bessere Fehlerbehebung und gezieltere Verbesserungen möglich wurden. Zum anderen konnten nun weitere Entwickler, auf einfache Weise, ihren Teil zu Redis beitragen und bei der Weiterentwicklung helfen.

Dabei bedeutet Open Source nicht zwingend, das jeder die Software frei nutzen und verändern kann. Um die jeweiligen Bedingungen möglichst klar festzulegen, kommen unterschiedliche Lizenzmodelle zum Einsatz, wobei für Redis die *3-Klausel-BSD-Lizenz* gewählt wurde[4]. Für die Nutzung von Redis in eigenen Softwareprodukten entstehen keine Kosten, sofern die jeweiligen Lizenz-Bedingungen eingehalten werden.

[1] Anwendung zur Analyse von Webseitenzugriffen

2.3 Aktuelle Entwicklung

Mit der redislabs Inc. hat sich letztlich auch ein kommerzieller Anbieter von Datenbanklösungen auf Basis von Redis gebildet. Dazu gehören bspw. Cloud-Installationen oder Datenbankcluster für Unternehmen. Zudem gehört redislabs zu den Hauptunterstützern des Projekts, sodass nun auch Salvatore Sanfilippo zu den Mitarbeitern des Unternehmens zählt[5]. Abseits davon arbeiten heute zahlreiche Entwickler an Verbesserungen und Fehlerbehebungen. Zudem haben sich neben Redis weitere neue Projekte gebildet, die bestimmte Gebiete abdecken. So ist mittlerweile eine graphische Oberfläche verfügbar und Portierungen auf weitere Betriebssysteme werden ebenso vorangetrieben, wobei die Weiterentwicklungen zum Großteil dem Open Source Schritt zu verdanken sind.

3. FUNKTIONSWEISE

3.1 NoSQL-Umsetzung

NoSQL-Datenbanken lassen sich grundsätzlich in vier verschiedene Gruppen einteilen. Redis gehört dabei zu den Vertretern der *Key-Value-Stores*[2], welche die geringste Komplexität aufweisen, da lediglich ein Schlüssel und ein Datenwert in der Datenbank abgelegt wird[6].

Wichtig ist hierbei vor allem, dass sich keine Abfragen nach den eigentlichen Werten durchführen lassen, sondern ausschließlich nach den Schlüsseln. Zudem werden an keiner Stelle Tabellen mit Schemata erstellt oder Datensätze untereinander verknüpft. Im Gegensatz zu anderen Datenbanken, bei denen die Bezeichnung NoSQL auch für *Not-Only-SQL* stehen kann, lässt Redis damit keinerlei Verwendung der Abfragesprache zu.

Zusätzlich wird Redis als *In-Memory-Datenbank* bezeichnet. Dies bedeutet, dass jegliche Daten durchgehend im Arbeitsspeicher des Servers gehalten werden. Hieraus resultiert auch ein wichtiger Faktor im Hinblick auf die Performance, da dadurch sehr schnelle Zugriffe möglich werden. Um den Betrieb auf Systemen mit geringer Hauptspeicherkapazität zu ermöglichen, existiert in den neuen Versionen auch die Möglichkeit zur Auslagerung auf einen virtuellen Speicher[7]. Letzterer wird dabei durch mindestens eine Festplatte bereitgestellt und enthält in der Regel die am wenigsten benötigten Schlüssel, um die Geschwindigkeit möglichst hoch zu halten.

3.2 Datenverwaltung

3.2.1 Grundlegendes

Obwohl das Konzept der Datenverwaltung sehr einfach ist, lassen sich auch umfangreichere Anwendungen mit Redis umsetzen. Dafür sorgt unter anderem, die bereits integrierte Unterstützung von fünf Datentypen, welche in nahezu jeder Programmiersprache vorkommen und sehr häufig verwendet werden. Allen voran gehören hierzu einfache Zeichenketten, aber ebenso Hashes, Listen, Mengen und sortierte Mengen, inklusive verschiedener Operationen.

Weitere Datentypen, wie Zahlenwerte oder Zeitangaben, werden mit Hilfe von Strings abgelegt. Es lassen sich aber ohne Weiteres mehrere Listen, Mengen oder Strings gemeinsam in einer Datenbank ablegen. Der jeweilige Entwickler sollte nur sicherstellen, die Schlüssel nach einem festen Schema zu vergeben, da diese für den späteren Zugriff auf die Daten unbedingt erforderlich sind. Für Schlüssel und Werte besteht zudem eine Maximalgröße von 512 MB, weshalb es z.B. möglich ist, eine komplette Bilddatei im String-Format in der Datenbank zu speichern. Von einer einfachen Benutzerzählung auf Webseiten, bis hin zur Ablage von Foto-Alben in einem sozialen Netzwerk, lassen sich also zahlreiche Anwendungsfälle umsetzen.

Im Übrigen grenzt sich Redis damit von weiteren bekannten Key-Value-Stores ab, wozu z.B. *Memcached* gehört, da diese oftmals nur einfache Strings unterstützen[8]. Daraus folgt mitunter der Vorteil, dass sich der Programmierer einer Anwendung, direkt auf die vorgefertigten Datentypen beziehen kann. Im Falle von Memcached müsste bereits vor der Ablage der Daten, eine vollständige Serialisierung stattfinden, um die Daten in ein geeignetes Format zu bringen. Da sich all diese Datenstrukturen generell nicht von denen anderer Programmiersprachen unterscheiden, wird im Folgenden nur auf einige Besonderheiten eingegangen, die in diesem Zusammenhang hervorzuheben sind.

3.2.2 Atomare Operationen

In fast allen Anwendungsszenarien müssen Clients auf eine Datenbank zugreifen, um Werte zu ändern und neu abzulegen. Dabei kann es vor allem dann zu Problemen kommen, wenn derselbe Wert von mehreren Clients gelesen, geändert und wieder übertragen wird, denn dadurch bleibt nur der zuletzt gespeicherte Wert erhalten. Dies ist im Allgemeinen auch unter dem Namen *Race Condition* bekannt.

Um dieses Fehlverhalten auszuschließen, wurden in Redis mehrere Funktionen integriert die dem vorbeugen. Wurde zu einem Schlüssel eine Zahl hinterlegt, lässt sich diese in nur einem Befehl inkrementieren und dekrementieren[9]. Der Client liest den Wert also nicht erst aus, sondern teilt dem Server in einem Schritt mit, um welchen Wert die Zahl erhöht werden soll. Bekannte Beispiele für diese Art von Datensätzen, wären ein Besucherzähler einer Webseite oder auch die Anzahl der aktuell geöffneten Serververbindungen.

Wird in einem komplexeren Kontext die Durchführung einer atomaren Operation benötigt, empfiehlt es sich die Befehle als *Transaktion* auszuführen[10]. Diese werden von Redis als logische Einheit betrachtet, sodass die komplette Abfolge der eingegebenen Befehle abgearbeitet wird, bevor ein anderer Client, die darin betroffenen Daten lesen oder ändern kann. Versuchen viele Clients gleichzeitig einen Datensatz zu ändern, kann das zwar einen Rückgang der Performance bedeuten, jedoch wird dafür die Datenkonsistenz gewahrt. Die Menge an Befehlen, die eine Transaktion umfasst, sollte aus diesem Grund so klein wie möglich gehalten werden.

3.2.3 Zeitlich beschränkte Daten

Als weitere Besonderheit bietet Redis die Möglichkeit, das Bestehen von Daten zeitlich einzuschränken. Dabei wird für einen bestimmten Schlüssel festgelegt, nach wie vielen Sekunden dieser gelöscht wird. Ob es sich dabei um eine kurze Zeitspanne von ein paar Minuten oder direkt um mehrere Wochen handelt, bleibt dem Programmierer überlassen. Im Nachhinein lässt sich durch gezielte Befehle abfragen, ob ein Wert mit einer Ablaufzeit versehen wurde, wie lang diese noch ist oder ob der Wert bereits gelöscht wurde. Zusätzlich kann die gespeicherte Ablaufzeit stetig erneuert werden.

[2] weitere sind spaltenorientierte, dokumentenorientierte und graphbasierte Datenbanken

Wichtig dabei ist allerdings, dass die Zeit auf der des Servers basiert, also z.B. des UNIX-Systems. Wird diese manuell geändert, könnten also Schlüssel gelöscht werden, die eigentlich noch existieren sollten. Selbst wenn der Redis-Dienst abgeschaltet wird, also die Datenbank außer Betrieb ist, wird beim nächsten Start die vergangene Zeit berechnet und eine entsprechende Löschung vorgenommen. Auf die gleiche Weise lässt sich statt einer Ablaufzeit, der genaue Zeitpunkt festlegen, zu dem der Datensatz gelöscht werden soll[11]. Diese Funktion sollte also insgesamt betrachtet, unbedingt mit Bedacht verwendet werden, um einen Datenverlust zu vermeiden, denn schon das manuelle Umstellen der Uhrzeit, könnte eine Löschung einzelner Datensätze zur Folge haben.

3.3 Persistente Speicherung

Da alle Daten durchgehend im Hauptspeicher des Datenbankservers gehalten werden, stellt sich natürlich die Frage, was im Falle eines Systemabsturzes passiert. Hierzu bietet Redis zwei verschiedene Betriebsmodi an, um einem Datenverlust entgegen zu wirken.

Bei der ersten Variante wird ein kompletter Datenbestand auf der Festplatte gesichert, der zum jeweiligen Zeitpunkt besteht. Dabei werden durch den Administrator, zwei Werte vorgegeben. Diese geben zum einen das Zeitintervall selbst an, z.B. alle 15 Minuten, zum anderen wird die Anzahl der Änderungen definiert, die mindestens erfolgt sein müssen[2]. Somit kann der Administrator flexibel entscheiden, wann die Sicherungen erstellt werden, denn es lassen sich ohne Weiteres mehrere dieser Regeln konfigurieren. Allerdings bringt dies zugleich einen Nachteil mit sich, denn eine garantierte Wiederherstellung der Daten ist nicht möglich. Sobald aktuelle Änderungen in der Datenbank bestehen, aber noch kein Sicherungsintervall erreicht wurde, wären diese bei einem Systemausfall nicht rekonstruierbar.

Alternativ lässt sich der *AOF-Modus (Appen-Only-File)* nutzen. Dieser veranlasst Redis bei Änderungen, den jeweiligen Schreibbefehl in einer Datei abzulegen. Je nach Wahl der Einstellung, wird diese Datei bei jeder stattfindenden Änderung oder jede Sekunde auf die Festplatte geschrieben. Zusätzlich kann man die Entscheidung an das Betriebssystem weitergeben. Sollte der Server daraufhin ausfallen, lässt sich dessen Zustand anhand der gesicherten Befehle wiederherstellen[8]. Im Hinblick auf die Performance hat dieser Modus verschiedene Auswirkungen. Wird bei jedem Schreibbefehl eine Sicherung durchgeführt, kann das den Server natürlich enorm belasten und verlangsamen. Jede Sekunde eine Sicherung durchzuführen, stellt daher einen guten Kompromiss dar. Überlässt man hingegen dem Betriebssystem die Wahl, lässt sich keine sichere Aussage darüber treffen, wie viele Daten wirklich verloren gegangen sind.

Vergleicht man die beiden Ansätze miteinander, lässt sich schnell auf die jeweiligen Folgen in der Praxis schließen. Beim ersten Modus wird der Server im laufenden Betrieb mehr belastet, hingegen ist im zweiten Fall die Wiederherstellung aus den Befehlen aufwendiger. Sollen alle gesicherten Daten nach einem Ausfall schnell wieder bereitstehen, ist daher die erste Variante zu empfehlen, welche auch die Standardeinstellung ist. Grundsätzlich gibt Redis allerdings keine Garantie darüber, alle Daten zu jeder Zeit wiederherstellen zu können, da beide Modi eine gewisse Fehlertoleranz und Inkonsistenz in Kauf nehmen.

3.4 Lastverteilung und Ausfallsicherheit

Sollte die Performance mit einem Redis Server nicht ausreichen oder möchte man ein möglichst ausfallsicheres System konzipieren, kann eine Replikation mit weiteren Systemen erfolgen, die im Allgemeinen auch als *Clustering* bezeichnet wird. Dazu unterstützt Redis die sogenannte *Master-Slave Konfiguration*, bei der die Slave-Systeme, welche ebenfalls Redis-Server sind, eine lokale Kopie des Datenbestands vom Master beziehen und diese den Clients im Lesemodus anbieten. In Figure 1 wird ein beispielhafter Aufbau einer solchen Konfiguration gezeigt.

Änderungen an den Daten werden in diesem Modus ausschließlich auf dem Master durchgeführt, wodurch jedes Mal ein Update in Echtzeit an den einzelnen Slave-Systemen ausgelöst wird. Sollte ein Client dennoch versuchen Daten auf einem Slave zu ändern, liefert dieser lediglich eine Fehlermeldung als Antwort. Das Abschalten dieser Fehlermeldung ist nicht empfehlenswert, da dies verschiedene Fehler provoziert[7]. Bei genauer Betrachtung, ist diese Art der Konfiguration nicht vollständig konsistent, da die Updates an den Clients asynchron durchgeführt werden. Das bedeutet, dass es durchaus vorkommen kann, das ein Client Daten eines Slave-Systems liest, diese aber am Master bereits geändert wurden. Ein Master teilt also einem Client nicht sofort mit, dass Änderungen bestehen und die entsprechenden Datensätze zum Lesen gesperrt werden müssen. Dies bietet zwar hinsichtlich der Leistung Vorteile, geht aber mit Kompromissen bei der Datenkonsistenz einher.

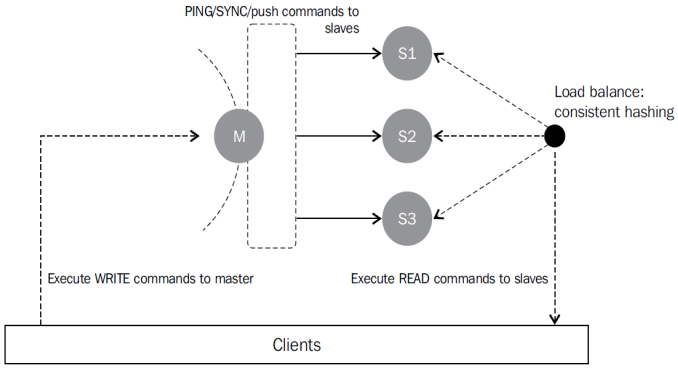

Figure 1: Replikations-Strategy Master-Slave[7]

Neben der Lastverteilung lässt sich die Konfiguration auch als Ausfallsicherheit verwenden, da sobald der Master nicht mehr erreichbar ist, ein angeschlossenes Slave-System dessen Arbeit übernehmen kann. Mit steigender Anzahl an Slave-Servern, entsteht jedoch ein Nachteil, denn der Master kann unter Umständen nicht mehr alle in ausreichender Geschwindigkeit mit den Datenbeständen versorgen. Dazu lässt sich eine Hierarchie von Servern aufbauen, in der es zu einem Slave, wiederum weitere Slave-Server geben kann. Der Master gibt also nur noch Änderungen an die Slave-Systeme weiter, die direkt unter ihm stehen. Von weiteren möglichen Servern hat er logisch gesehen keine Kenntnis.

Kommen Schreibvorgänge ebenso häufig vor, wie Leseanfragen, bietet es sich an einen *Master-Master-Cluster* mit Redis umzusetzen. Dabei wird der Datenbestand auf verschiedene Master-Server aufgeteilt, sodass die Schreibvorgänge ebenfalls möglichst gleichmäßig, an allen Servern stattfinden. Diese Konfiguration wird von Redis jedoch nicht un-

terstützt und muss vom Programmierer der Anwendung implementiert werden, weshalb sie hier nicht näher betrachtet wird.

4. PRAKTISCHER EINSATZ

4.1 Installation und Einrichtung

Für die Nutzung von Redis sind zwei grundlegende Komponenten nötig. Zum einen wird ein Server benötigt, auf dem die eigentliche Datenbank betrieben wird, zum anderen sind Clients für den Zugriff erforderlich. Bei der Serveranwendung zählen vor allem Linux- und Unix-Distributionen zu den unterstützten Betriebssystemen. Dabei ist die Installation schon mit wenigen Befehlen in der Konsole erledigt[12].

Windows wird offiziell zwar nicht unterstützt, jedoch arbeitet die Microsoft Open Tech Group an einer Portierung zum 64-Bit Windows Betriebssystem. Diese wird durchgehend gepflegt, sodass auch dort ein Einsatz der Datenbank möglich wird[13].

Nach der Installation bietet Redis eine integrierte Kommandozeile, die zur Kommunikation mit dem Server dient. Über diese lassen sich bereits alle Funktionen verwenden, begonnen bei der Ablage der ersten Daten, bis hin zu speziellen Anpassungen des Servers. Jeder eingegebene Befehl, wird mit einer entsprechenden Antwort vom Server quittiert, sodass Fehler direkt ersichtlich sind.

Wem die Nutzung der Konsole nicht zusagt, kann alternativ den Redis Desktop Manager verwenden. Dieser lässt sich unter Linux, Windows und Mac OS X installieren und stellt im Anschluss eine Verbindung zum gewünschten Server her[14].

4.2 Programmierschnittstellen

Soll Redis in eigenen Projekten verwendet werden, kann zwischen zahlreichen Schnittstellen, für die verschiedensten Programmiersprachen, gewählt werden. Zu den bekanntesten Sprachen gehören dabei Perl, PHP, C/C++, sowie Python und Java. Alleine für letztere gibt es dabei schon mehrere verschiedene Implementierungen, die zum Teil sehr spezielle Ansätze haben[15]. Dabei handelt es sich bspw. um einfache graphische Oberflächen oder eine Unterstützung für bestimmte Java Versionen. Zur schnellen Anbindung eines Redis Servers und weitere vielfältige Aufgaben, eignet sich unter anderem *Jedis*. Das folgt nicht zuletzt aus der Unterstützung für die aktuelle Version 3.0.x und der einfachen Verwendung[16]. Um einen Redis Server anzusprechen und mit dem ersten Schlüssel und Wert zu versorgen, reichen schon die folgenden Codezeilen aus.

```
Jedis jedis = new Jedis("Serveradresse");
jedis.set("Schlüssel", "Wert");
```

Um nun einen Schritt weiterzugehen, lässt sich das Beispiel um folgende Befehle ergänzen.

```
Transaction t = jedis.multi();
t.hset(1,"user","admin","pw","root");
t.hset(2,"user","guest","pw","free");
t.exec();
String username = jedis.hget("1", "user");
```

Im vorstehenden Code wird zu Beginn eine Transaktion definiert und durch die Methode *multi* eingeleitet. Anschließend werden zwei Hashes hinzugefügt, die einen Nutzernamen und das Passwort enthalten. Durch den Aufruf von *exec* wird die Transaktion letztlich gestartet und am Ende wird der *user* zum Schlüssel *1* abgefragt. Eine Leseanfrage zum Schlüssel *2*, würde zu einer Fehlermeldung führen, solange die Transaktion läuft.

4.3 Nachrichtendienst

Eine Eigenschaft die Redis bereits in der Grundinstallation mitbringt, betrifft den Nachrichtendienst, der kurz mit *pub-sub (Publisher-Subscriber)* bezeichnet wird. Mit diesem lässt sich relativ schnell ein einfacher Dienst programmieren, der z.B. als Kommentarfunktion in einer Webanwendung agiert. Hierzu wird, wie in Figure 2 dargestellt, von einem Publisher ein Schlüssel erzeugt, zu dem eine Nachricht hinterlegt wird, wobei man den Schlüssel in diesem Zusammenhang als *Kanal (Channel)* bezeichnet[11]. Jeder Client mit Zugriff auf den Server, kann diesem Kanal als Subscriber beitreten, woraufhin dauerhaft jede neue Nachricht direkt an ihn weitergeleitet wird. Dabei kann der Client natürlich jederzeit entscheiden die Verbindung zu beenden.

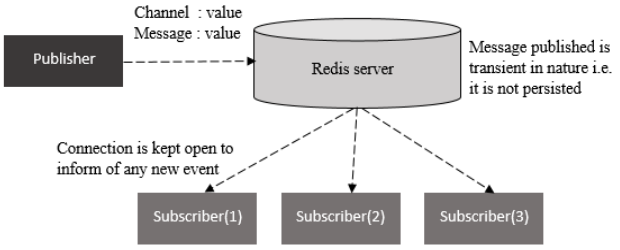

Figure 2: Publisher versendet Nachricht an alle Subscriber[7]

Dieses Modell ähnelt im Grundsatz denen von sozialen Netzwerken, weshalb sich Redis gerade in diesem Bereich besonders eignet.

4.4 Scripts

Seit der Version 2.6 unterstützt Redis die Script-Funktion, welche vor allem zur Erweiterung dienen soll, ohne dabei den Quellcode ändern zu müssen. Dazu wurde die Sprache *Lua* gewählt, die insbesondere bei der Entwicklung von Spielen eingesetzt wird. Zur Übertragung des Scripts zum Server wird das String-Format verwendet und jedes Script wird vollständig vom Server abgearbeitet, sobald er es erhält. Da währenddessen der Server blockiert wird, sollte ein Script immer für spezielle Aufgaben genutzt werden und möglichst schnell auszuführen sein. Neben dem Erstellen und Abfragen von Schlüsseln, kann auch ein Wert vom Script zurückgegeben werden, der sich dann für weitere Funktionen nutzen lässt[2]. Grundsätzlich lassen sich Lua-Scripts als kleine Programme ansehen, die für spezielle Anwendungsfälle dienen. Dabei weisen sie eine Ähnlichkeit zu *JavaScript* auf, weshalb sich auch ähnliche Aufgaben mit ihnen durchführen lassen. Sofern bestimmte Vorgänge immer wieder abgearbeitet werden müssen, sind Lua-Scripts die ideale Lösung. Dabei kann es sich z.B. um ein Script handeln, das durch verschiedene Abfragen, die aktuelle Belastung des Servers zusammenfasst und das Ergebnis in einer Datei sichert. Anhand dieser könnte der Administrator eine Analyse durchführen, ob der Server noch Leistungskapazitäten besitzt.

4.5 Leistung

Der Hauptvorteil von Redis liegt in der schnellen Verarbeitung einfacher Daten. Damit sich jeder Programmierer einer Anwendung, ein möglichst genaues Bild von der wirklichen Performance machen kann, haben die Entwickler ein Script zur Leistungsmessung bereitgestellt[17]. Dies läuft auf der jeweils eigenen Hardware und kann mit verschiedenen Werten gestartet werden, die unter anderem angeben wie viele Clients, Zugriffe und Befehle simuliert werden sollen. Im Anschluss lässt sich daran gut abschätzen, zu welcher letztlichen Geschwindigkeit das eigene System mit installiertem Redis Server fähig wäre.

Zum Testen wurde das Script auf einem einfachen Notebook ausgeführt, das einen Intel Core 2 Duo Prozessor und 4 GB Arbeitsspeicher besitzt. Als Testparameter wurden nur die set- und get-Operationen, zum Speichern und Lesen von Schlüsseln definiert. Zudem wurden 50 Clients und 100.000 Anfragen simuliert, wobei es sich um die Standardeinstellungen handelt. Das Ergebnis sieht man nach der Ausführung auf der Redis-Konsole.

```
SET: 37174.72 requests per second
GET: 37313.43 requests per second
```

Somit wären bereits auf einem solchen System, rund 37.000 Anfragen in der Sekunde möglich, wodurch ersichtlich wird, warum Redis in den entsprechenden Bereichen einen solchen Erfolg hat. Dabei wurde hier lediglich eine einfache Konfiguration und Basishardware gewählt, die mit weiteren Anpassungen noch optimierbar ist. Allerdings ist es im Zweifelsfall ratsam, eine eigene Überprüfung durchzuführen, wenn klar ist welche Datentypen verwendet werden sollen. Insgesamt bleibt dies zwar ein Benchmark, der lediglich eine Simulation der Realität durchführt, als erster Anhaltspunkt ist dies aber durchaus brauchbar.

5. FAZIT

Wie in der vorliegenden Ausarbeitung zu lesen, handelt es sich bei Redis um einen sehr intelligenten und ausgereiften Key-Value-Store, der seine Vorteile insbesondere durch die einfache Verwendung ausspielen kann. Dass nach Möglichkeit alle Daten im Arbeitsspeicher des Serversystems gehalten werden, beschleunigt die Zugriffe enorm und bietet gerade gegenüber herkömmlichen Datenbanken, wie MySQL, Vorteile. Mit Hilfe der Implementierung verschiedener Datentypen, kann sich Redis aber ebenso gegen andere NoSQL-Datenbanken im direkten Vergleich durchsetzen. Die Unterstützung für ein breites Spektrum an Programmiersprachen und die Möglichkeit der hohen Skalierung, lassen Redis insgesamt zu einem ernstzunehmenden Datenbankverwaltungssystem werden. Dem Einsatz in Unternehmensanwendungen steht, nach einer genauen Prüfung der jeweiligen Anforderungen, also nichts im Wege.

Dank der Veröffentlichung des Quellcodes und der regen Beteiligung vieler Entwickler, entstehen stetig neue interessante Projekte, bei denen Redis entweder um bestimmte Eigenschaften erweitert oder aber in neue Umgebungen portiert wird. Da sich eine Anwendung im produktiven Betrieb nur schwer auf eine neue Datenbasis umziehen lässt, stellt auch die Lizenzierung von Redis einen Vorteil dar. Denn durch die 3-Klausel-BSD-Lizenz kann praktisch jeder, der eine Software programmiert und Redis zur Verwaltung in Betracht zieht, einen Testlauf durchführen. Im Zweifelsfall spart dies hohe Kosten, die im späteren Verlauf entstehen könnten. Die zahlreichen Funktionen, die Redis bereits enthält, wie den Nachrichtendienst, ersparen unter Umständen sogar Programmieraufwand.

Jedoch sollte immer beachtet werden, das Redis auch einige Nachteile mit sich bringt. So werden nur grundlegende Datentypen unterstützt und sehr komplexe Datenstrukturen, lassen sich nur schwer oder gar nicht abbilden. Daher wird Redis als NoSQL-Datenbank keineswegs die bisher genutzten relationalen Datenbanken vollständig ersetzen können, sondern lediglich als Ergänzung dienen. Wer zudem auf eine uneingeschränkte Datenkonsistenz und Speicherung angewiesen ist, sollte Redis nicht einsetzen, da diese, aufgrund der gegebenen Funktionsweise, nicht garantiert werden können.

6. REFERENCES

[1] Salvatore Sanfilippo. Lloogg realtime web log analyzer. https://github.com/antirez/lloogg, Abrufdatum 20.11.2015.

[2] Maxwell Dayvson Da Silva and Hugo Lopes Tavares. *Redis Essentials*. Packt Publishing Ltd., Birmingham, 2015.

[3] Salvatore Sanfilippo. Redis - offizielle webseite. http://redis.io/, Abrufdatum 20.11.2015.

[4] Open Source Initiative. The bsd 3-clause license. https://opensource.org/licenses/BSD-3-Clause, Abrufdatum 12.12.2015.

[5] Inc. Redis Labs. redislabs - home of redis. https://redislabs.com/, Abrufdatum 20.11.2015.

[6] Dan McCreary and Ann Kelly. *Making Sense of NoSQL*. Manning Publications Co., Shelter Island, 2014.

[7] Vinoo Das. *Learning Redis*. Packt Publishing Ltd., Birmingham, 2015.

[8] Josiah L. Carlson. *Redis in Action*. Manning Publications Co., Shelter Island, 2013.

[9] Tiago Macedo and Fred Oliveira. *Redis Cookbook*. O'Reilly Production Services, Sebastopol, 2011.

[10] Gleicon Moraes, Dov Murik, and Matti Paksula. The redis cookbook. http://www.rediscookbook.org/get_and_delete.html, Abrufdatum 27.11.2015.

[11] Eric Redmond and Jim R. Wilson. *Seven Databases in Seven Weeks*. Pragmatic Programmers, LLC., Dallas, Texas, 2012.

[12] Salvatore Sanfilippo. Redis - download. http://redis.io/download, Abrufdatum 22.11.2015.

[13] Microsoft Open Technologies. Open tech project redis. https://msopentech.com/opentech-projects/redis/, Abrufdatum 20.11.2015.

[14] Igor Malinovskiy. Redis desktop manager. http://redisdesktop.com/, Abrufdatum 05.12.2015.

[15] Salvatore Sanfilippo. Redis - clients. http://redis.io/clients, Abrufdatum 23.11.2015.

[16] Jonathan Leibiusky. Jedis. https://github.com/xetorthio/jedis, Abrufdatum 07.12.2015.

[17] Salvatore Sanfilippo. Redis - benchmarks. http://redis.io/topics/benchmarks, Abrufdatum 13.12.2015.