Vergleich von NoSQL Datenbanksystemen im Anwendungsfall einer verteilten Sessionkontext-Datenbank

Marius Gerling
UNIBERG GmbH - 23816 Bebensee
Hochschule Darmstadt - Fachbereich Informatik - 64295 Darmstadt
Email: marius.gerling@uniberg.com

Die Auswahl einer Datenbanktechnologie und eines Datenbankproduktes für einen bestimmten Anwendungsfall ist eine schwierige und langwierige Angelegenheit. Mit diesem Dokument soll ein Vergleich von verschiedenen Datenbankprodukten für den Anwendungsfall einer verteilten Sessionkontext-Datenbank dargestellt werden. Unter den getesteten Datenbankprodukten befinden sich Vertreter verschiedener Datenbanktechnologien. Der Anwendungsfall der verteilten Sessionkontext-Datenbank steht dabei stellvertretend für den einfachen Zugriff auf Key-Value Daten in einer verteilten Umgebung. Für die Auswertung des Produktvergleiches wird nicht nur die Antwortzeit und die Ressourcennutzung des Produktes bewertet, sondern es wird auch besonders auf Ausfallszenarien in der verteilten Datenbankumgebung eingegangen.

In diesem Dokument wird zunächst eine detaillierte Beschreibung des Anwendungsfalles einer verteilten Sessionkontext-Datenbank gegeben. Ebenfalls wird im ersten Kapitel der Aufbau der Testplattform beschrieben. Kapitel zwei betrachtet die Auswahl verschiedener Datenbankprodukte, welche in diesem Dokument betrachtet werden. Das dritte Kapitel befasst sich mit den verschiedenen Testszenarien die eine Datenbank absolvieren muss. Im Anschluss werden die Messdaten in Kapitel vier ausgewertet, so dass im Kapitel fünf eine Auswahl für ein Datenbankprodukt getroffen und ein Fazit gezogen werden kann.

Durch die Verwendung einer virtuellen Testumgebung ohne garantierte Performance der einzelnen Maschinen ist das Ergebnis der Evaluation nicht ohne Einschränkungen zu verwenden. Generell kann aber ein Trend erkannt und für die Entscheidung der Verwendung eines Datenbankproduktes oder Datenbanktypes verwendet werden.

1 Einleitung

1.1 Anwendungsfall Sessionkontext-Datenbank

Der Anwendungsfall Sessionkontext-Datenbank beschreibt die Verwendung einer Datenbank für das Halten von session-bezogenen Daten. Diese Daten fallen in diversen Anwendungen an, welche Sessions verwenden. Das wohl bekannteste Beispiel einer Session-basierten Anwendung ist eine Webseite mit Login-Funktionalität wie Facebook[1].

Ein Sessionkontext beschreibt dabei die Daten, welche die Anwendung für eine Session vorhält. Dies können zum Beispiel Einstellungen eines Benutzers sein oder die letzten Interaktionen, welche auf der Webseite durchgeführt worden sind. Ebenso wird der Status der Session gesichert.

[1] `https://facebook.com`

Im hier simulierten Anwendungsfall wird von einer einheitlichen Größe der Sessionkontexte von 1,5 Kilobyte[2] ausgegangen. Dabei wird im unterschiedlichen Maß auf die Sessionkontexte zugegriffen oder diese verändert. Das folgende Abbild 1 zeigt die zugehörige Zustandsmaschine, welche eine einzelne Session in der Simulation durchläuft. Rechteckige Felder zeigen dabei an, dass ein Zugriff auf die Datenbank durchgeführt wird.

Jede Session wird dabei zu Beginn Ihrer Laufzeit angelegt und in der Datenbank persistiert (CreateSession). Sobald dies abgeschlossen ist, wird eine zufällige Zeit gewartet. Im Durchschnitt sind dies 300 Sekunden. Im Anschluss wird der Sessionkontext aus der Datenbank gelesen (GetSession). Danach wird anhand des Zufalles eine gewichtete Entscheidung zwischen den drei nächsten möglichen Zuständen getroffen. Zu 50% ist der nächste Zustand wiederum das Warten auf einen erneuten Zugriff auf die Session. In diesem Falle wurde der Sessionkontext nur gelesen. Mit einer Wahrscheinlichkeit von 45% werden die Sessioninformationen aktualisiert und in die Datenbank geschrieben (UpdateSession). Zuletzt wird mit einer Wahrscheinlichkeit von 5% eine Session beendet (DeleteSession). Wird eine Session beendet wird diese aus der Datenbank entfernt und es wird direkt im Anschluss eine neue Session gestartet und in die Datenbank gespeichert. Die Last bleibt in diesem Anwendungsfall immer gleich.

Insgesamt werden im Testaufbau 900.000 Sessions simuliert. Bei der Verteilung der Anfragen bedeutet dies eine Anfragelast von 4.650 Anfragen pro Sekunde gegen die Datenbank. Diese teilen sich in 1650 schreibende und 3000 lesende Anfragen auf.

$$\frac{Anfragen}{s} = \frac{900.000}{300s} * (\frac{50+3*45+3*5}{100}) = \frac{4.650}{s} \quad (1)$$

[2]Die Sessiongröße geht aus dem Anwendungsfall eines Kunden hervor

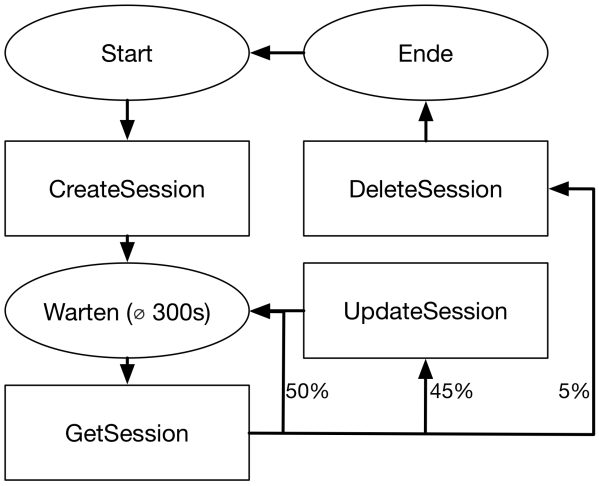

Abbildung 1. Zustandsmaschine einer Sessions

1.2 Problembeschreibung und Lösungsansatz

Die Sessionkontexte werden aktuell in einer In-Memory-Datenbank lokal am Standort vorgehalten. Geht man von einer größeren Webseite aus, welche mehrere Rechenzentren für die Anfrageverarbeitung verwendet, kann dies zu einem Problem werden. Zum einen muss eine Session seine Anfragen immer an den gleichen Standort senden, um den Sessionkontext verwenden zu können. Zum anderen gibt es hier Probleme in der Sessionverarbeitung, wenn ein Rechenzentrum ausfällt oder aus der Anfrageverarbeitung genommen wird. Dies hat zur Folge, dass die Anfragen von einem anderen Rechenzentrum verarbeitet werden, welches den Sessionkontext nicht kennt. Eine neue Session muss aufgebaut werden. Die Nutzungserfahrung des Kunden wurde beeinträchtigt.

Mit Hilfe einer verteilten Datenbanklösung soll dem Problem der 1:1 Zuordnung einer Session zu einem Rechnenzentrum entgegengewirkt werden. Mit einer verteilten Datenbank soll der Sessionkontext an allen verfügbaren Rechenzentren vorgehalten und zeitnah aktualisiert werden. Eine Zuordnung einer Session zu einem Standort kann dabei nicht aufgelöst werden, jedoch gehen bei einem Standortausfall oder der Abschaltung eines Standortes (z.B. für Wartungsarbeiten) nur ein Bruchteil der Sessions verloren.

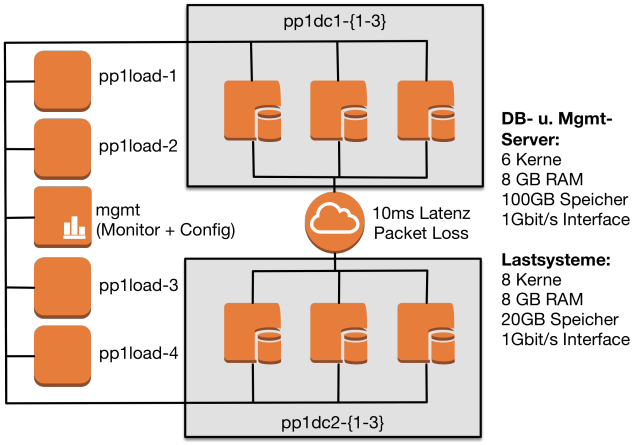

Abbildung 2. Hardware- und Netzplan

1.3 Aufbau der Testplattform

In diesem Kapitel wird die Testplattform sowohl im Netzaufbau wie auch im logischer Aufbau der Testanwendungen und Auswertesystemen beschrieben.

1.3.1 Hardware- und Netzaufbau

Die Testplattform wurde in der Openstack[3]-Umgebung der UNIBERG GmbH virtualisiert aufgebaut. Insgesamt wurden dazu elf Systeme aufgesetzt. Darunter fallen sechs Datenbanksysteme, vier Lastsysteme sowie ein Managementsystem. In Abbildung 2 ist der Netzaufbau der Umgebung dargestellt.

Zu beachten ist dabei, dass die Datenbankmaschinen nur mittels latenz- und paketverlustbehafteter Verbindungen miteinander sprechen können. Dies simuliert den Betrieb der verteilten Datenbank in zwei Rechenzentren. Mittels des Kernelmoduls netem[4] wird auf der Weitverkehrsverbindung zwischen den simulierten Rechenzentren eine Latenz von 10 (+-1)ms sowie ein Packet-Loss von 0.01% eingerichtet.

Da es sich bei dem Aufbau der Umgebung um eine virtuelle Umgebung handelt und keine garantierte Leistung für die Performance der einzelnen Maschinen gegeben werden konnte, ist die Durchführung der Tests nicht repräsentativ. Mit Hilfe der in diesem Dokument zusammengetragenen Ergebnisse kann jedoch ein Trend erkannt werden, da jede der getesteten Datenbanken mit den gleichen Vorraussetzungen kämpfen musste.

1.3.2 Applikationsaufbau

Um den Anwendungsfall auf Basis der konfigurierten Testplattform testen zu können, wurde eine Testapplikation entwickelt sowie ein Framework für das Ermitteln, Zusammentragen und Visualisieren der Messwerte verwendet. Auf jedem der vier Lastsysteme werden dazu je 15 Instanzen der Testapplikation mit jeweils 15.000 simulierten Sessions gestartet. Jede Testapplikation wählt alle 20 Millisekunden eine zufällige Session aus, welche die Zustandsmaschine[5] durchläuft. Jede Applikationsinstanz generiert dabei 77,5 Anfragen an die Datenbank pro Sekunde. Für jede der Anfragen an eine Datenbank werden dabei die Antwortzeiten in ein Histogramm gespeichert sowie die Anzahl der der gesamten Anfragen aber auch der fehlgeschlagenen Anfragen erfasst. Diese Werte werden aus der Applikation heraus an ein lokales Prometheus[6] PushGateway gesendet, welches die Metriken vorhält.

Der zentrale Managementserver besitzt eine Prometheus Instanz, welche die Informationen von den PushGateway-Instanzen auf den Lastsystemen einsammelt und speichert. Zur Visualisierung der Metriken wird Grafana[7] verwendet. Ebenfalls werden die Systemwerte (CPU-Last, RAM-Belegung, IO-Raten, ...) von allen Maschinen mittels des Prometheus Node-Exporters erfasst und ebenfalls in Prometheus gesammelt und mittels Grafana visualisiert. Dies Ermöglicht auch eine Aussage zur Last der Systeme zu treffen.

Der Applikationsaufbau ist in der folgenden Abbildung 3 dargestellt.

[3] `https://openstack.org`
[4] `https://wiki.linuxfoundation.org/networking/netem`
[5] Siehe Abbildung 1
[6] `https://prometheus.io`
[7] `https://grafana.com`

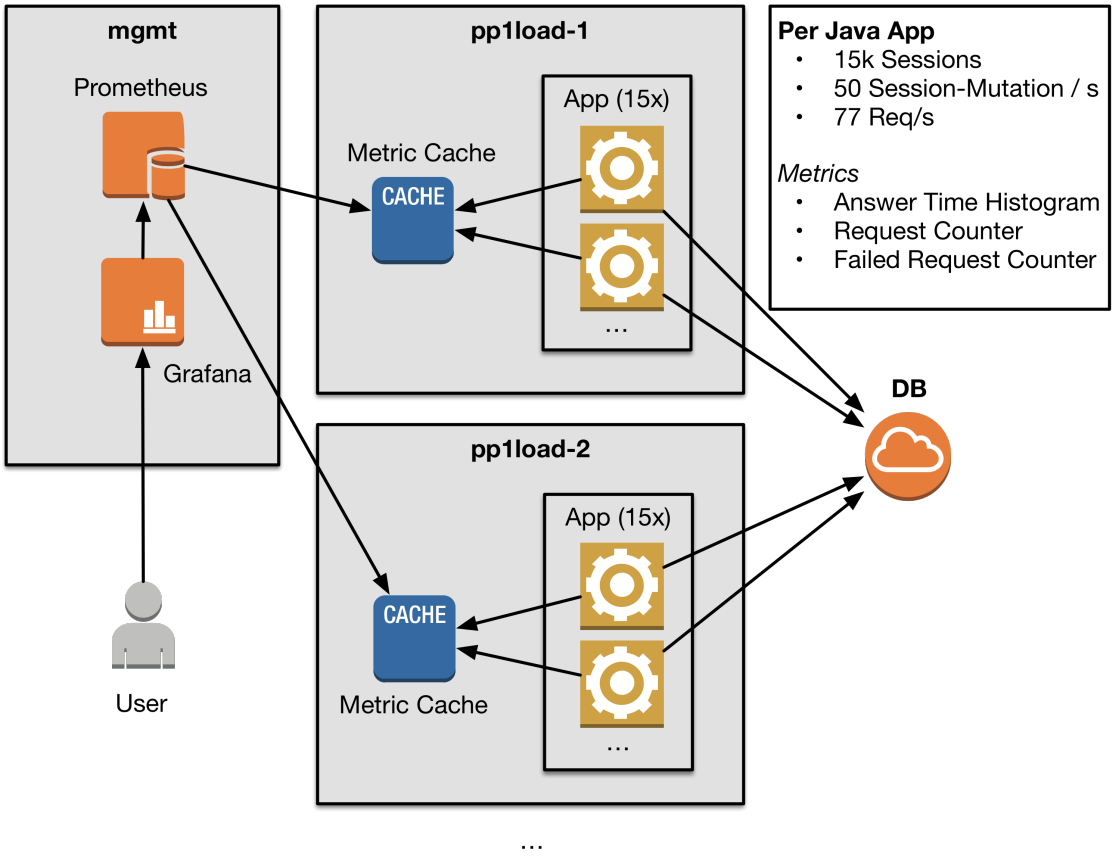

Abbildung 3. Logischer Applikationsaufbau

1.4 Anforderungen an das Datenbanksystem

Um die Tauglichkeit eines Datenbanksystems für den Einsatz einer Sessionkontext-Datenbank zu ermitteln, wurden einige Anforderungen gestellt, welche jedes getestete Datenbanksystem einhalten soll.

Für die Auswertung der Performance wurden folgende Anforderungen gestellt. Die Perzentile beziehen sich dabei immer auf Basis einer Minute.

* 50% Perzentil der Antwortzeiten <= 10ms
* 90% Perzentil der Antwortzeiten <= 20ms
* 99% Perzentil der Antwortzeiten <= 100ms
* 99.9% Perzentil der Antwortzeiten <= 200ms

Neben den Performanceanforderungen werden auch Anforderungen an die Persistenz der Daten an den verschiedenen Standorten gestellt. Bei einem Rechenzentrumsausfall sollen maximal ein Promille der Sessions auf Fehler laufen. Es dürfen also 900 Sessions beeinträchtigt werden. Aufgrund des homogenen Lastbildes von 1650 schreibenden Anfragen pro Sekunde bedeutet dies, dass die Latenz der Datenübertragung maximal 546ms betragen darf. Die Applikation prüft dazu die Inhalte der Sessiondaten. Ebenfalls muss die Datenbank einen automatischen Schwenk bei einem Server- bzw. Rechenzentrumsausfall unterstützen.

$$SyncLatenz = \frac{1}{(\frac{1650 wReq/s}{900})} \approx 546ms \qquad (2)$$

2 Marktanalyse der NoSQL-Datenbankprodukte

Um eine Auswahl an NoSQL-Datenbankprodukten zu erstellen, werden zunächst funktionale Anforderungen an das Datenbankprodukt gestellt. Im Anschluss werden Datenbankprodukte gesucht, welche den Anforderungen entsprechen. Um die Anzahl der Datenbankprodukte zu begrenzen, werden ähnliche Datenbankprodukte wie Duplikate behandelt. Es wird nur eines dieser Produkte getestet.

2.1 Anforderungen an das Datenbankprodukt

Bei der Marktanalyse werden die folgende Anforderungen an das Datenbankprodukt gestellt. Zunächst muss das Datenbankprodukt eine stark verteilte Architektur verwenden. Damit fallen relationale ACID-Datenbanken bereits aus.[8] Das Zusammenspiel der Konsistenz (*Consistency*), Verfügbarkeit (*Availability*) und Partitionstoleranz (*Partition Tolerance*) ist dabei im CAP-Theorem beschrieben.[9] Zwei der drei Ziele können erreicht werden, wobei das dritte nur mit einem Best-Effort Ansatz erfüllt wird, jedoch nicht garantiert werden kann. Aufgrund der Tatsache, dass Inkonsistenzen im Use-Case in Kauf genommen werden, ist die Partitionstoleranz und die Verfügbarkeit (AP) sehr wichtig.[10]

Neben der hohen Verfügbarkeit und der Partitionstoleranz ist ein schneller Zugriff auf einzelne Datensätze nötig, um die Performanceanforderungen erfüllen zu können.

2.2 Datenbankprodukte

Für die Auswahl der Datenbankprodukte wurde die Grafik 4 auf der folgenden Seite verwendet. Für den Einsatzzweck einer Sessionkontext-Datenbank ist hier vor allem der linke Abschnitt „Fast Lookups“ sinnvoll, da über einen Key auf einzelne Sessionkontext-Daten zugegriffen wird.

Redis Cluster[11] wird als In-Memory-Datenbank verwendet. Redis Cluster ist jedoch nicht partitionstolerant. Es ist jedoch die aktuell verwendete Technologie für diesen Anwendungsfall und dient daher auch als Referenz für die Antwortzeiten.

Als alternative Datenbanken werden Cassandra[12] und RIAK[13] betrachtet. Diese basieren auf dem Dynamo Paper [DHJ+07], welches eine AP-Datenbankarchitektur beschreibt.

Cassandra wird verwendet, weil es die quelloffene Referenzimplementierung dieses Paper ist und oft eingesetzt wird. Das Interessante an RIAK ist, dass diese Datenbank auch mit einem In-Memory Backend verwendet werden kann. Dadurch sind bessere Antwortzeiten möglich.

Um die Partitionstoleranz[14] von RIAK verwenden zu können, wird jedoch eine Premium-Version benötigt, welche für den Test nicht genutzt werden konnte, da Basho, die Firma hinter RIAK, Insolvenz angemeldet hat. Eine andere Firma übernimmt den Quellcode und wird auch die Premiumfunktionen in die quelloffene Version übernehmen. Dies wird jedoch erst nach der Veröffentlichung dieses Dokumentes geschehen.[15]

Als Ersatz für RIAK wird Couchbase[16] verwendet. Dies hat mit der Verwendung der XDCR-Funktionalität ebenfalls die Möglichkeit eine AP-Datenbank darzustellen.[17]

3 Testszenarien

In diesem Kapitel werden die Testszenarien beschrieben, welche die unterschiedlichen Datenbankprodukte durchlaufen müssen. Anhand der hier ermittelten Messwerte werden die Fähigkeiten der Datenbanken ausgewertet.

3.1 Antwortzeiten im Normalbetrieb

Im Normalbetriebstest wird das Antwortzeitenverhalten der Datenbanksysteme über einen Zeitraum von 24 Stunden beobachtet. Dafür werden zunächst alle Sessions aufgebaut. Im Anschluss durchlaufen 3000 Sessions pro Sekunde die Zustandsmaschine. Die Antwortzeiten der 4650 Anfragen pro Sekunde werden dabei ausgewertet.

Während der Ausführung dieses Tests sind alle Datenbankserver verfügbar und können Anfragen beantworten.

---

[8] [MK16, S. 135ff]
[9] [GL02]
[10] [MK16, S. 135ff]
[11] https://redis.io/topics/cluster-spec
[12] http://cassandra.apache.org
[13] http://basho.com/products/riak-kv/
[14] [GL02]
[15] https://groups.google.com/forum/#!topic/riak-users/eX3n8Hv6e0c
[16] https://www.couchbase.com
[17] https://blog.couchbase.com/cap-theorem-and-couchbase-server-time-\xdcr/

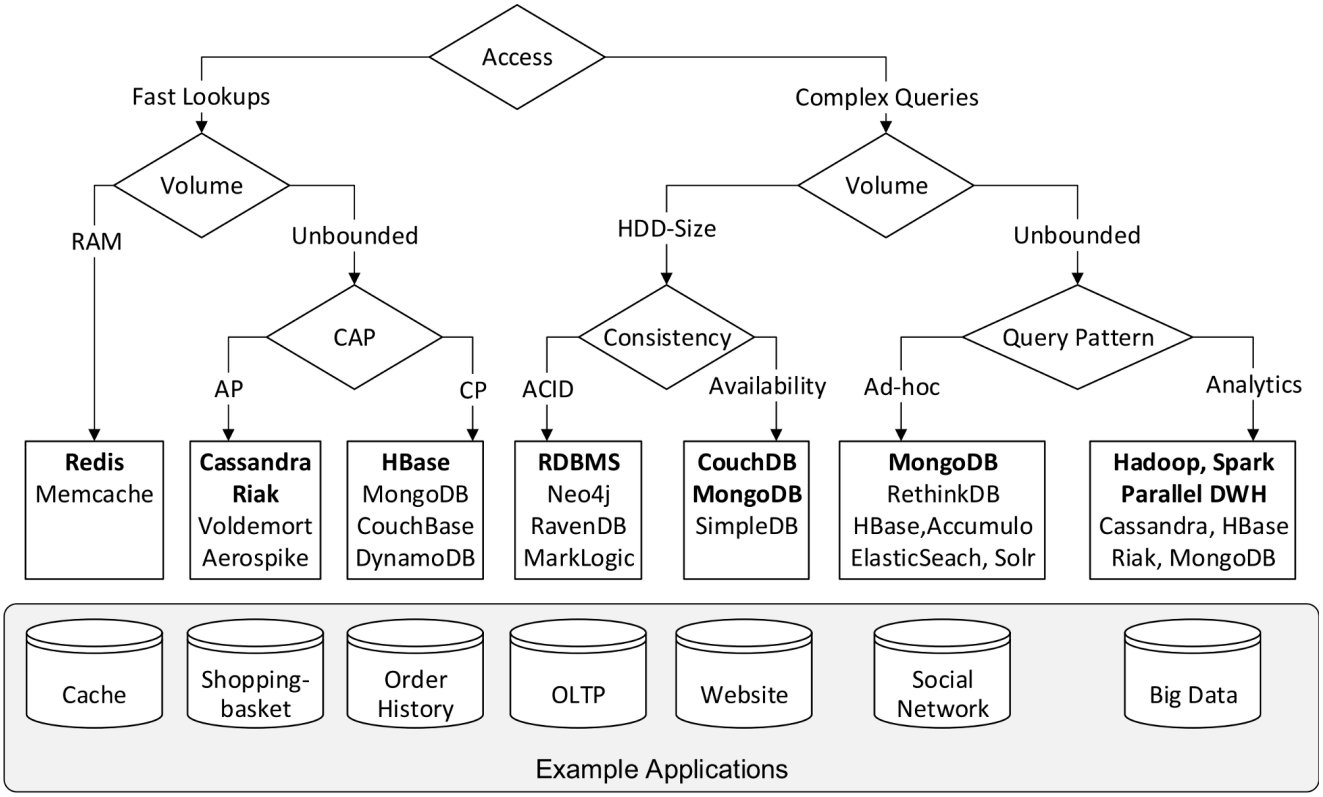

Abbildung 4. Entscheidungsbaum zur Auswahl von NoSQL Datenbanksystemen [Ges16]

3.2 Wartung eines Servers

Bei dem Test Wartung eines Servers wird das Verhalten des Datenbanksystems beobachtet, wenn ein Datenbankserver ausgeschaltet wird. Dies soll die Wartung eines Datenbankservers simulieren. Bevor der Datenbankserver ausgeschaltet wird, wird der Normalbetrieb für eine halbe Stunde ausgeführt. Es wird davon ausgegangen, dass es dabei zu keinen fehlerhaften Anfragen kommt. Eine Erhöhung der Antwortzeit wird in diesem Szenario toleriert.

3.3 Ausfall eines Servers

Neben dem Wartungstest wird ebenfalls der Ausfall eines Servers getestet. Der Unterschied zum Wartungstest ist, dass hier die Datenbankinstanz auf dem Server nicht heruntergefahren wird, sondern der Datenbankprozess mittels „SIGKILL“[18] beendet wird.

3.4 Ausfall eines Rechenzentrums

Der Test Ausfall eines Rechenzentrums testet das Verhalten der Datenbank während eines der beiden Rechenzentren ausfällt. Dies wird getestet indem alle Datenbankprozesse eines Rechenzentrums gleichzeitig terminiert werden. Das verbleibende Rechenzentrum muss die gesamte Anfragelast verarbeiten. In diesem Szenario ist ebenfalls mit fehlerhaften Anfragen sowie veralteten Zustandsdaten zu rechnen.

4 Auswertung der Datenbankprodukte

In diesem Kapitel werden die Tests mit den verschiedenen Datenpankprodukten ausgewertet. Ebenso werden hier die unterschiedlichen Eigenschaften der Datenbanken aufgelistet, welche bei der Einrichtung, Nutzung und Betrieb der verteilten Datenbank festgestellt wurden.

[18] `http://man7.org/linux/man-pages/man7/signal.7.html`

Es wird eine Bewertung der Datenbanktechnologien mit bis zu fünf Punkten in den Kategorien Antwortzeiten im Normalbetrieb, Wartung eines Servers, Ausfall eines Servers und Ausfall eines Rechenzentrums durchgeführt. Die Datenbanktechnologie, welche die meisten Punkte erhält, eignet sich am Besten für den Einsatz im definierten Anwendungsfall.

4.1 Redis Cluster

Als erstes Datenbankprodukt wird Redis Cluster bewertet. Dieses ist die Referenzimplementierung für die Performance im Anwendungsfall, da es das Datenbankprodukt ist, welches derzeit eingesetzt wird.

4.1.1 Einrichtung, Nutzung und Betrieb

Eine einzelne Redis Instanz lässt sich einfach einrichten. Die Datenverteilung muss jedoch anhand von zwei Parametern eingerichtet werden. Zum einen werden die Daten von Redis auf mehrere Hosts verteilt (Sharding). Ebenso muss die Verfügbarkeit dieser Datenteile durch mehrere Hosts gewährleistet werden (Replikation). Seit 2015[19] bietet Redis mit Redis Cluster[20] die Möglichkeit diese Features von Haus aus zu nutzen.

Redis Cluster ist jedoch keine wirkliche verteilte Datenbank. Es gibt immer einen Master, welcher Schreibzugriffe für einen Datensatz erlaubt. Die Anzahl der Master ist dabei die Anzahl der Shards, auf denen die Daten verteilt werden. Ein Masterausfall kann durch den Cluster automatisch beseitigt werden, wenn das Replika des Masters verfügbar ist und mehr als 50% aller Master noch verfügbar sind. In diesem Fall übernimmt das Replika (Slave) die neue Masterrolle.

Schreibzugriff auf einen Datensatz ist nur bei einem Master des jeweiligen Datensatzes möglich. Diese Einschränkung sorgt dafür, dass die verteilte Nutzung über mehrere Rechenzentren nicht möglich ist. Ein Client muss z.B. für Schreibzugriffe auf den Master des Keys in einem anderen Rechenzentrum zugreifen. Die Abbildung 5 zeigt die Verteilung des Datenbestands über drei

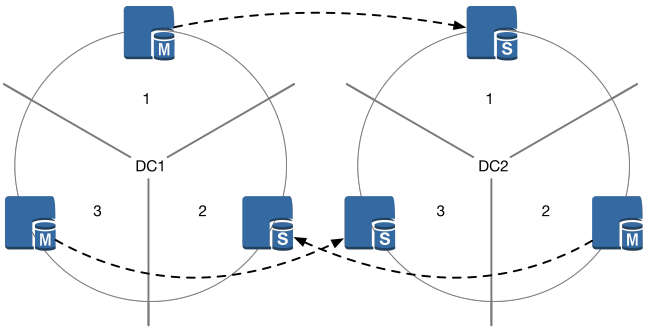

Abbildung 5. Redis Cluster Aufbau (Master und Slave)

Master und jeweils ein Slave für jeden Datenteil, welcher auf einem Slave liegt. Die Master verteilen sich dabei über beide Rechenzentren. Die gestrichelten Linien zeigen den Replikationsverkehr zwischen den Datenbankinstanzen in unteschiedlichen Rechenzentren.

4.1.2 Antwortzeiten im Normalbetrieb

In diesem Abschnitt werden die Antwortzeiten der Datenbank im Normalbetrieb ausgewertet. Ein zeitlicher Ablauf der Antwortzeiten über 24 Stunden ist in Abbild 8 (S. 12) zu finden. Zu sehen ist, dass die Antwortzeiten im 50% bis 99% Perzentil über die gesamte Zeit stabil verlaufen. Lediglich das 99,9% Perzentil zeigt einige Ausreißer nach oben. Insgesamt lässt sich sagen, dass 99% aller Anfragen in unter 11 Millisekunden beantwortet werden. Im Schnitt werden ebenfalls 99,9% der Anfragen in unter 11 Millisekunden beantwortet.

Dies erfüllt die gestellten Anforderungen vollumfänglich. Damit werden fünf von fünf Punkten für Redis vergeben.

4.1.3 Wartung eines Servers

Die Wartung eines Redis Datenbankservers ist gleichzusetzen mit dem Ausfall eines Servers, da die Datenbank selbst keine Stopp-Routine bereitstellt.

[19] https://groups.google.com/forum/#!msg/redis-db/dO0bFyD_THQ/Uoo2GjIx6qgJ

[20] https://redis.io/topics/cluster-spec

4.1.4 Ausfall eines Servers

Der Ausfall eines Redis Datenbankservers hat keinerlei Auswirkungen auf die Anfrageverarbeitung. Sollte der ausgefallene Server ein Master gewesen sein, übernimmt der Slave die Anfrageverarbeitung nahtlos. Dies ist möglich, da die Mehrheit der Master noch zur Verfügung steht. Der Ausfall eines Servers wird daher mit fünf von fünf Punkten bewertet.

4.1.5 Ausfall eines Rechenzentrums

Ein Rechenzentrumsausfall ist im hier vorgestellten Testaufbau schwierig zu bewerten. Es kann dabei zu einem Totalausfall der Datenbank kommen. Dies ist der Fall, wenn die Mehrheit der Master ausfällt. Da kein „Quorum“[21] erreicht wird, können die Replika nicht automatisch aktiviert werden.

Um dies zu erreichen ist eine externe Kontrollinstanz notwendig, welche die „Leader Election“[22] durchführt und die Replika zu Mastern befördert. Eine mögliche Implementierung für diese Aufgabe ist das Programm Apache Zookeeper[23], welches oft im Zusammenspiel mit Redis eingesetzt wird.

Ein Test mit einer solchen externen Kontrollinstanz wurde in dieser Arbeit nicht durchgeführt. Der Ausfall eines Rechenzentrums wird mit einem von fünf Punkten bewertet.

4.2 Cassandra

Als zweite Datenbank wird Cassandra getestet. Die Datenbank zeichnet sich durch Ihre vielfältigen Anpassungsmöglichkeiten für die Verteilung des Datenbestandes und für die Datenbankzugriffe aus.

4.2.1 Einrichtung, Nutzung und Betrieb

Da Cassandra sich in seinen Eigenschaften individuell konfigurieren lässt, konnte der Plattformaufbau mit Cassandra detailliert geplant werden. Um eine lokale Ausfallsicherheit in einem Rechenzentrum gewährleisten zu können, wird sowohl bei Lese- als auch bei Schreibzugriffen

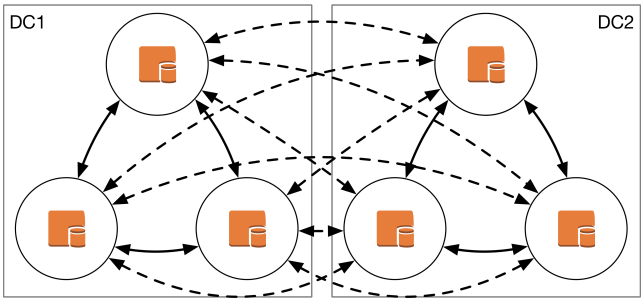

Abbildung 6. Cassandra Cluster Aufbau & Kommunikationsbeziehungen

das „Consistency-Level“ des Zugriffes auf „Local Quorum“ gesetzt. Um eine Ausfallsicherheit von einem Server gewährleisten zu können, müssen daher drei Server den Datensatz enthalten. Ebenso soll der Datensatz in mehreren Rechenzentren verfügbar sein. Dies sorgt dafür, dass jeder der sechs Datenbankserver alle Datensätze hält.[24]

Cassandra verwendet dabei bei Anfragen an den Datenbankcluster immer die zwei Datenbankserver, welche aktuell für diesen Client am schnellsten Antworten. Gleichzeitig werden die Schreibanfragen bereits an weitere Datenbankserver gesandt. Wird die Anfrage von zwei Datenbankservern erfolgreich durchgeführt ist das Local Quorum erreicht und die Abfrage gilt als erfolgreich durchgeführt.

In Abbildung 6 ist der Aufbau des Datenbankclusters mitsamt den Kommunikationsbeziehungen der einzelnen Server zu sehen. Die durchgezogenen Linien stellen dabei rechenzentrumslokalen Replikationsverkehr und gestrichelte Linien rechenzentrumsübergreifenden Replikationsverkehr dar. Dabei hält jeder Server ein Abbild aller Daten. Cassandra ist dabei in einer „Masterless“ und „Shared-Nothing“ Architektur aufgebaut.[25]

4.2.2 Antwortzeiten im Normalbetrieb

Die Antwortzeiten im Normalbetrieb sind ebenso wie die Antwortzeiten des Redis Tests dargestellt und sind in Abbild 9 (S. 12) einsehbar. Es ist zu erkennen, dass die Antwortzeiten nicht so stabil verlaufen wie die Antwortzeiten der Redis Datenbank. Im Graphen verhalten sich die Linien

[21] [OP11, S. 450f]
[22] [Ray13, S. 77ff]
[23] `https://zookeeper.apache.org/`

[24] [CH16, S. 133ff]
[25] [Sto86]

nicht so stabil auf einer Höhe, wie sie es bei Redis getan haben.

Es ist zu erkennen, dass die Antwortzeiten bis zum 95% Perzentil besser sind als die der Redis Datenbank (6ms vs. 10ms). Jedoch werden die Antwortzeiten beim 99 und auch beim 99,9% Perzentil immer schlechter. (99%: 23ms vs. 10ms ; 99,9%: 77ms vs 10ms)

Die Latenzanforderungen werden eingehalten. Diese sind jedoch schlechter als die Antwortzeiten von Redis. Es werden vier von fünf Punkten für Cassandra vergeben.

4.2.3 Wartung eines Servers

Da Cassandra ebenfalls keine Stopp-Routine bereitstellt, ist dies im Test gleichzusetzen mit dem Ausfall eines Servers.

4.2.4 Ausfall eines Servers

Der Ausfall eines Servers hat keinerlei Auswirkungen auf die Anfrageverarbeitung. Es kommt zu keinen fehlerhaften Anfragen gegen die Datenbank. Auf basis des Local-Quorum wird immer der aktuellste Datensatz aus dem vorliegenden Rechenzentrum verwendet.

Cassandra wird in diesem Test mit fünf von fünf Punkten bewertet.

4.2.5 Ausfall eines Rechenzentrums

Bei dem Ausfall eines Rechenzentrums kommt es ebenfalls zu keinen fehlerhaften Anfragen. Ein Wechsel auf das andere Rechenzentrum funktioniert fehlerfrei.

Ebenfalls wurden keine veralteten Daten im Ausweichrechenzentrum gelesen. Dies hängt damit zusammen, dass sogar der Client die Datenbankanfragen an weitere Server aus dem anderen Rechenzentrum sendet. Dies geschieht jedoch asynchron, so dass die Antwortzeiten davon nicht beeinträchtigt werden.[26]

Cassandra wird in diesem Test mit fünf von fünf Punkten bewertet.

[26] Siehe `http://docs.datastax.com/en/developer/java-driver/3.3/manual/load_balancing/#dc-aware-round-robin-policy`

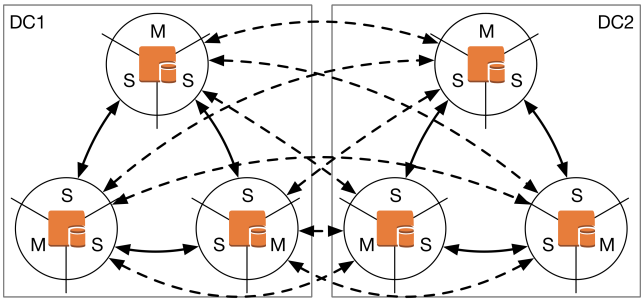

Abbildung 7. Couchbase Cluster Aufbau & Kommunikationsbeziehungen

4.3 Couchbase

Zuletzt wird Couchbase getestet. Diese Datenbank verspricht dank verteilter In-Memory Datenhaltung ähnliche Performance wie Redis zu liefern, gleichzeitig jedoch eine größere Ausfallsicherheit zu bieten.

4.3.1 Einrichtung, Nutzung und Betrieb

Couchbase hat eine ähnliche Architektur der Datenhaltung wie Cassandra. Auch hier wird auf jedem Server jedes Datum gehalten. Ebenso werden die Daten zu jedem Server im anderen Rechenzentrum repliziert.

Ein gravierender Unterschied zu der Architektur von Cassandra ist, dass es hier lokale Master zu einem Datensatz gibt. Ein Datensatz kann nur auf einer für ihn zuständigen Masterinstanz geschrieben werden. Die anderen Server in diesem Rechenzentrum sind jeweils Slaves bzw. Replika für diese Datensätze.

Der Aufbau des Couchbase Clusters mitsamt den Kommunikationsbeziehungen ist im Bild 7 (S. 12) abgebildet. Die durchgezogenen Linien stellen dabei rechenzentrumslokalen Replikationsverkehr und gestrichelte Linien rechenzentrumsübergreifenden Replikationsverkehr dar.

4.3.2 Antwortzeiten im Normalbetrieb

Die Antwortzeiten im Normalbetrieb sind als Graph in Abbildung 10 dargestellt. Im Vergleich zum Referenzprodukt Redis sind die Antwortzeiten bis zum 99% Perzentil gleich gut. Die Antwortzeiten für das 99,9% Perzentil sind jedoch mit 20ms doppelt so groß wie die Antwortzeiten bei Redis. Die Antwortzeiten sind damit sehr gut und werden mit fünf von fünf Punkten bewertet.

4.3.3 Wartung eines Servers

Die Wartung eines Servers ist möglich, jedoch mit erheblichem Aufwand verbunden. Zunächst muss der Couchbase Server aus dem lokalen Cluster ausgegliedert werden. Dieser Prozess sorgt dafür, dass die zugewiesenen Mastersektoren auf die verbleibenden Server übertragen werden. Ist dieser Prozess abgeschlossen, kann der Server für die Wartung freigestellt werden. Die Wartung eines Servers wird für Couchbase mit fünf von fünf Punkten bewertet.

4.3.4 Ausfall eines Servers

Der Ausfall eines Servers sorgt in Couchbase für den logischen Wegfall eines gesamten Rechenzentrums. Da die Masterinstanz für einige Datenstäze nicht mehr vorhanden ist, können an diesem Standort keine schreibenden Zugriffe für diesen Datensatz durchgeführt werden. Ein Schwenk zum alternativen Standort ist nötig.

Der Ausfall eines Servers wird mit drei von fünf Punkten bewertet.

4.3.5 Ausfall eines Rechenzentrums

Der Ausfall eines Rechenzentrums sorgt im Fall von Couchbase zu einem Schwenk der Clients auf das andere Rechenzentrum. Dies funktioniert ohne Probleme. Jedoch kann es bei Couchbase zu veralteten Datensätzen in dem anderen Rechenzentrum kommen. Während der Tests wurden hier unterschiedliche Werte zwischen null und fünf veralteten Datensätzen ermittelt. Dies liegt voll im Soll von bis zu 900 veralteten Datensätzen, ist jedoch nicht so gut wie das Verhalten von Cassandra.

Beim Wiederanlauf eines Standortes sollte man bei Couchbase die Replikation der anderen Standorte zunächst abschließen lassen, bevor wieder Datenzugriffe zugelassen werden. Während der Tests kam es zu der Situation, dass die Applikation schon wieder Zugriffe auf die Datenbank getätigt hat, während im Hintergrund noch eine Replikation Daten zurückgespielt hat. Diese Daten haben die neueren von der Applikation geschriebenen Daten wiederum überschrieben. Couchbase scheint hier nicht mit „Vector clocks“[27] zu arbeiten. Man sollte darauf achten erst auf ein Rechenzentrum zuzugreifen, wenn dieses die Replikation auf dem aktuellen Stand ist.

Der Ausfall eines Rechenzentrums wird für Couchbase mit vier von fünf Punkten bewertet.

Tabelle 1. Auswertungsergebnis

Test	Redis	Cassandra	Couchbase
Normalbetrieb	5	4	5
Wartung	5	5	5
Serverausfall	5	5	3
RZ-Ausfall	1	5	4
Gesamt	16	19	17

5 Ergebnis und Fazit

Nachdem die verschiedenen Datenbankprodukte getestet und die Tests ausgewertet wurden, kann ein Ergebnis gezogen werden. Dazu sind die erreichten Punkte in den verschiedenen Tests in Tabelle 1 aufgeführt.

Obwohl Cassandra bei der Performance zurückfällt, konnte es dieses Defizit durch die starke Ausfallsicherheit wieder gut machen. Insgesamt erzielt Cassandra mit 19 von 20 möglichen Punkten das beste Ergebnis und geht damit als Gewinner aus der Evaluation der Datenbanktechnologien hervor.

Doch auch die anderen Datenbanktechnologien schlagen sich nicht schlecht in dieser Auswertung und taugen generell für diesen Anwendungsfall. Vor allem ist dies der Fall, wenn die Performance eine sehr große Rolle spielt. Ebenfalls scheint Riak ein interessantes Datenbankprodukt zu sein, welches jedoch nicht in dieser Arbeit getestet werden konnte.

Die Testapplikation und der Testaufbau ist in einem Git Repository unter `https://github.com/uniberg/distributed-key-value-db-evaluation` öffentlich zugänglich.

[27] [Fid88]

Literatur

[CH16] CARPETER, J ; HEWITT, E: *Cassandra: The Definitive Guide*. 2. Aufl. Sebastopol : O'Reilly, 2016. – ISBN 978–1–491–93366–4

[DHJ+07] DECANDIA, G ; HASTORUN, D ; JAMPANI, M ; KAKULAPATI, G ; LAKSHMAN, A ; PILCHIN, A ; SIVASUBRAMANIAN, S ; VOSSHALL, P ; VOGELS, W: *Dynamo: Amazon's Highly Available Key-value Store*. http://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf, 2007. – Abgerufen am 10. September 2017

[Fid88] FIDGE, C J.: Timestamps in Message-Passing Systems That Preserve the Partial Ordering. In: *Australian Computer Science Communications* Vol. 10/ No. 1 (1988), S. 56–66

[Ges16] GESSERT, F: *NoSQL Databases: a Survey and Decision Guidance*. https://medium.baqend.com/nosql-databases-a-survey-\and-decision-guidance-\ea7823a822d, 2016. – Abgerufen am 10. September 2017

[GL02] GILBERTF, S ; LYNCH, N: Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. In: *ACM SIGACT News* 33(2)/2014 (2002), S. 51–59

[MK16] MEIER, A ; KAUFMANN, M: *SQL- & NoSQL-Datenbanken*. 8. Aufl. Berlin : eXamen.press - Springer Vieweg, 2016. – ISBN 978–3–662–47664–2

[OP11] ÖZSU, M T. ; P, Valduriez: *Principles of Distributed Database Systems*. 3. Aufl. Berlin : Springer, 2011. – ISBN 978–1–4419–8834–8

[Ray13] RAYNAL, M: *Distributed Algorithms for Message-Passing Systems*. 1. Aufl. Berlin : Springer, 2013. – ISBN 978–3–642–38123–2

[Sto86] STONEBAKER, M: The Case for Shared Nothing. In: *Berkley* (1986)

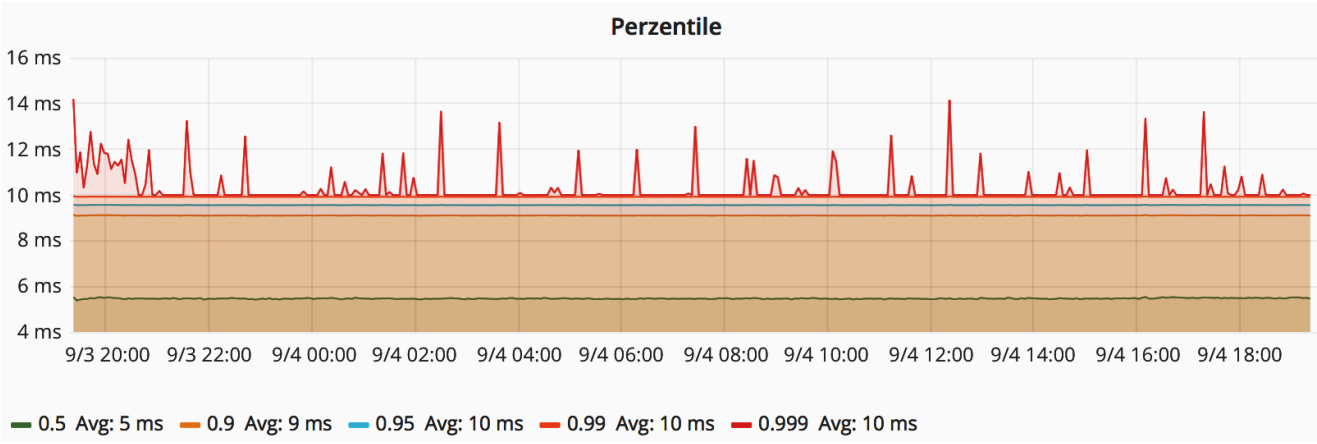

Abbildung 8. Antwortzeiten von Redis im Normalbetrieb

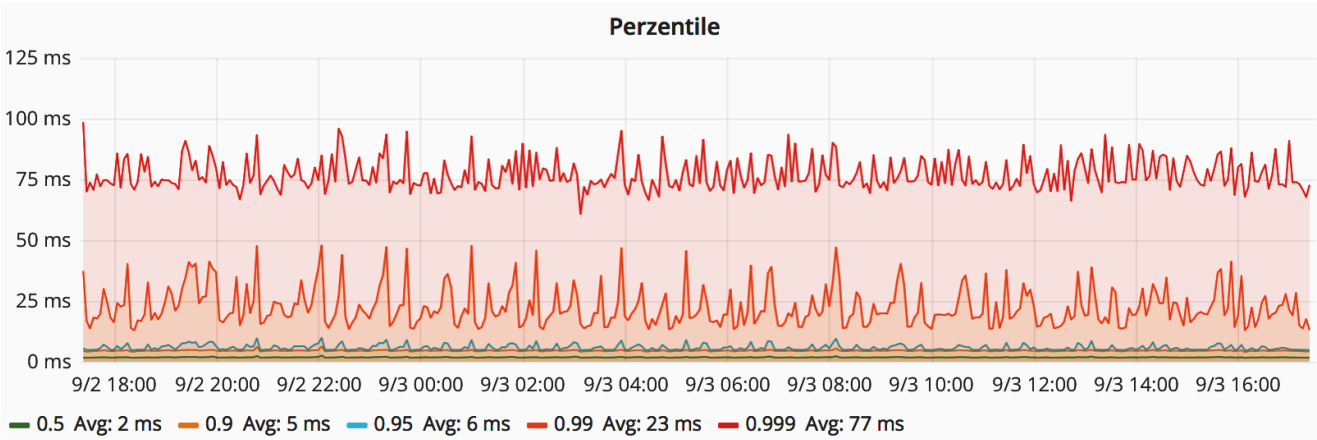

Abbildung 9. Antwortzeiten von Cassandra im Normalbetrieb

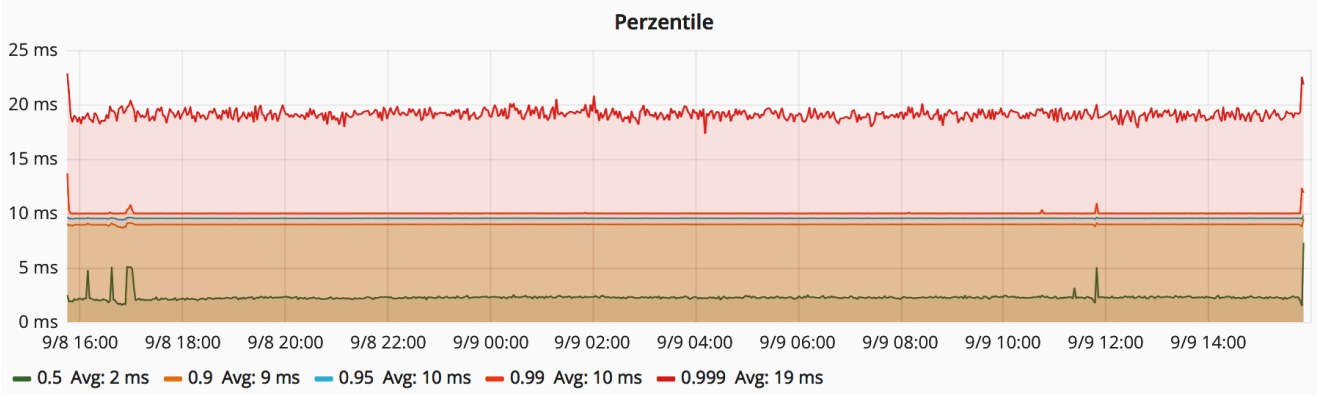

Abbildung 10. Antwortzeiten von Couchbase im Normalbetrieb