4 Replikation

In MongoDB wird die Replikation[1] zur Sicherstellung der Ausfallsicherheit (das P aus dem CAP-Theorem) und zur Skalierung von Lesezugriffen eingesetzt.

Dabei wird grob zwischen zwei Ansätzen unterschieden: dem reinen *Master-Slave*-Verfahren und dem sogenannten *Replica Set*, einem Master-Slave-Betrieb mit automatischem Failover. Die Begriffe Master-Slave und Replica Set werden wir in diesem Kapitel genauer definieren.

4.1 Das Oplog

Als Grundlage der Replikation dient das sogenannte *Oplog* (Kurzform von *Operations Log*). Dabei handelt es sich um einen Ringpuffer fester Größe, in dem alle schreibenden Operationen festgehalten werden.

Technisch ist das Oplog als Capped Collection realisiert, die im Falle eines Replica Set `local.oplog.rs` heißt. Nach einem Insert

```
rs1:PRIMARY> use test
switched to db test
rs1:PRIMARY> db.docs.insert({_id:1})
```

findet sich im Oplog dazu auch ein entsprechender Eintrag:

```
rs1:PRIMARY> use local
switched to db local
rs1:PRIMARY> db.oplog.rs.findOne({op:"i"})
{
        "ts" : Timestamp(1384517822, 1),
        "h" : NumberLong("-8300730980553944076"),
        "v" : 2,
        "op" : "i",
        "ns" : "test.docs",
        "o" : {
                "_id" : 1
        }
}
```

1. *http://de.wikipedia.org/wiki/Replikation_(Datenverarbeitung)*

Das Feld `op` gibt Auskunft über die Art der Operation (`i`=Insert, `u`=Update, `d`=Delete), `ns` über die betroffene Collection, und `o` enthält das Dokument, das eingefügt wurde bzw. die Änderungen enthält. Bei Updates werden dabei nicht die verändernden *Operationen* übertragen, sondern die aus der Operation resultierenden neuen *Werte* der betroffenen Felder. So wird die Idempotenz[2] der Oplog-Einträge sichergestellt. Aus einem inkrementellen Update

```
rs1:PRIMARY> use test
switched to db test
rs1:PRIMARY> db.docs.insert({_id:2, i:1})
rs1:PRIMARY> db.docs.update({_id:2}, {$inc: {i:1}})
```

wird im Oplog ein `$set` mit dem Zielwert `2`:

```
rs1:PRIMARY> use local
switched to db local
rs1:PRIMARY> db.oplog.rs.findOne({op:"u"})
{
        "ts" : Timestamp(1384758121, 1),
        "h" : NumberLong("-3252250995679452247"),
        "v" : 2,
        "op" : "u",
        "ns" : "test.docs",
        "o2" : {
                "_id" : 2
        },
        "o" : {
                "$set" : {
                        "i" : 2
                }
        }
}
```

Das Feld `o2` enthält dabei den Primärschlüssel des zu ändernden Dokuments. Beachten Sie, dass das Oplog ein internes Schema verwendet, das sich ggf. mit der Version des Servers ändern kann.

Allgemeine Informationen zum Status des Oplogs rufen Sie ab mit:

```
rs1:PRIMARY> db.printReplicationInfo()
configured oplog size:   50MB
log length start to end: 538secs (0.15hrs)
oplog first event time:  Mon Nov 04 2013 07:39:22 GMT+0100 (MEZ)
oplog last event time:   Mon Nov 04 2013 07:48:20 GMT+0100 (MEZ)
now:                     Mon Nov 04 2013 08:05:30 GMT+0100 (MEZ)
```

2. *http://de.wikipedia.org/wiki/Idempotenz*

Trigger im Eigenbau

Wenn Sie Mechanismen wie Trigger auf Ihren relationalen Tabellen vermissen, können Sie diese mithilfe eines sogenannten *Trailing Cursors* und des Oplogs realisieren. Ein Trailing Cursor kann allgemein auf Capped Collections verwendet werden. Er wartet fortlaufend auf neue, einzufügende Dokumente. Wenn Sie so das Oplog beobachten, können Sie Inserts, Updates usw. auf allen Collections abgreifen. Beispiele dafür gibt es etwa in Java[a] oder Node.js[b]. Beachten Sie aber dabei, dass sich das Schema der Oplog-Einträge ggf. mit neuen Versionen ändern kann.

a. *https://github.com/mongodb/mongo-java-driver/blob/master/examples/ReadOplog.java*
b. *https://github.com/mongodb/node-mongodb-native/blob/1.4/examples/oplog.js*

Kapazitätsplanung

Per Default beträgt die Größe des Oplogs auf 64-Bit-Systemen 5% des verfügbaren Plattenplatzes, mindestens aber 1 GB. Dieses Verhalten können Sie mit der Kommandozeilenoption `--oplogSize <MB>` ändern. In den folgenden Fällen kann es nötig sein, dass Sie eine größeres Oplog konfigurieren müssen:

- Updates, die oft viele Dokumente ändern. Zur Sicherstellung der Idempotenz der Oplog-Operationen werden aus einem Multi-Update mehrere Einträge im Oplog.
- Einfüge- und Löschoperationen halten sich in etwa die Waage. In diesem Fall wächst nicht so sehr die Größe der Datenbasis selbst, wohl aber die der aufzuzeichnenden Operationen.
- Ganz allgemein führen Updates, die kein Verschieben von Dokumenten auslösen, dazu, dass die Datenbasis sich nicht signifikant vergrößert, wohl aber viele Einträge im Oplog erzeugt.

4.2 Master-Slave-Replikation

Die Master-Slave-Replikation, bei der einzelne Server-Instanzen feste Rollen einnehmen, ist mittlerweile veraltet, soll aber der Vollständigkeit halber vorgestellt werden. Möglicherweise haben Sie einen Anwendungsfall, den Sie gut mit einer Master-Slave-Replikation lösen können, z. B. die Replikation von Daten zwischen mehreren physikalisch getrennten Rechenzentren und niedrigen Anforderungen an die Verfügbarkeit des Systems bei Schreibzugriffen.

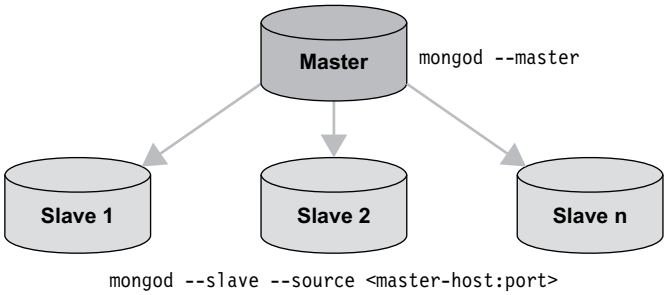

Abb. 4–1 *Master-Slave-Betrieb*

Wenn Sie die Master-Slave-Replikation verwenden, nimmt genau ein Server die Rolle des Masters ein. Der Master ist alleine für schreibende Zugriffe zuständig (Inserts, Updates und Removes) und repliziert diese Operationen an alle bekannten Slaves. Für das Starten des Masters sind folgende Kommandozeilenoptionen relevant:

Option	Bedeutung
--master	Startet die Server-Instanz als Master.
--oplogSize <MB>	Gibt die Größe des *Oplogs* (in Megabyte) an. Als Default werden 5% des verfügbaren Plattenplatzes verwendet.

Tab. 4–1 *Kommandozeilenoptionen für den Master*

Starten wir nun eine Instanz als Master, legen aber zuvor noch ein separates Verzeichnis für die Datenbankdateien an:

```
$ rem Windows
$ mkdir \data\master
$ # Unix
$mkdir -p /data/master
$ mongod --master --dbpath /data/master --oplogSize 50
```

Im Server-Log sehen wir, dass das Oplog angelegt wird, was sonst nicht der Fall ist:

```
Fri Oct 25 10:12:21.993 [initandlisten] MongoDB starting : pid=1456 port=27017
dbpath=/data/master master=1 64-bit host=
trelle2
Fri Oct 25 10:12:21.994 [initandlisten] db version v2.x
Fri Oct 25 10:12:21.994 [initandlisten] git version:
b9925db5eac369d77a3a5f5d98a145eaaacd9673
...
Fri Oct 25 10:12:22.179 [initandlisten] ******
Fri Oct 25 10:12:22.179 [initandlisten] creating replication oplog of size:
50MB...
Fri Oct 25 10:12:22.596 [initandlisten] ******
...
```

Das Oplog wird in der Capped Collection `local.$oplog` gespeichert, wie ein Blick in die Mongo Shell verrät:

```
$ mongo local
MongoDB shell version: 2.x
connecting to: local
> show collections
oplog.$main
startup_log
```

Die Datenbank `local` *ist von der Replikation ausgeschlossen.*

Zum Starten eines Slaves sind diese Kommandozeilenoptionen von Bedeutung:

Option	Bedeutung
--slave	Startet die Server-Instanz als Master.
-- source <master-server:port>	Name und Port der Master-Instanz.
--only <db>	Schränkt die Replikation dieses Slaves auf eine Datenbank namens <db> ein.
--slavedelay <sec>	Definiert eine Verzögerung von <sec> Sekunden, mit der die Operationen des Masters auf den Slave angewandt werden sollen.
--autoresync	Führt automatisch eine Resynchronisierung durch, falls die Daten des Slaves veraltet sind.

Tab. 4–2 *Kommandozeilenoptionen für die Slaves*

Richten wir nun zur Demonstration zwei Slaves ein, von denen der zweite selektiv nur die Datenbank `geheim` replizieren soll.

```
mkdir -p /data/slave0
mkdir -p /data/slave1
mongod --slave --source localhost:27017 --dbpath /data/slave0 --port 9000
mongod --slave --source localhost:27017 --dbpath /data/slave1 --port 9001 --
only geheim
```

Danach verbinden wir uns mit dem Master und finden die beiden Slaves in der Collection `local.slaves` wieder:

```
$ mongo local
MongoDB shell version: 2.x
connecting to: local
> db.slaves.find().pretty()
{
        "_id" : ObjectId("52661141f8b998e01a13185a"),
        "config" : {
                "host" : "127.0.0.1:55446",
                "upgradeNeeded" : true
        },
        "ns" : "local.oplog.$main",
        "syncedTo" : Timestamp(1382694168, 1)
}
```

```
{
        "_id" : ObjectId("526a3d4cec380ff8855ea911"),
        "config" : {
                "host" : "127.0.0.1:55527",
                "upgradeNeeded" : true
        },
        "ns" : "local.oplog.$main",
        "syncedTo" : Timestamp(1382694698, 1)
}
```

Jetzt schreiben wir auf dem Master ein Dokument in die Datenbank `test`:

```
> use test
switched to db test
> db.buecher.insert({text: "Onkel Toms Hütte"})
> exit
```

Werfen Sie nun einen Blick auf die Log-Ausgaben der Slave-Prozesse, die die Daten nun replizieren, die Datenbank `test` anlegen usw. Wir überprüfen anschließend, ob unsere Daten auch auf den Slaves gelandet sind:

```
$ mongo --port 9000 --eval db.buecher.find().forEach(printjson)
MongoDB shell version: 2.x
connecting to: 127.0.0.1:9000/test
{ "_id" : ObjectId("526a3fb7b40735d56376dae7"), "text" : "Onkel Toms Hütte" }
```

Auf dem zweiten Slave finden wir nichts, da dieser nur die Datenbank `geheim` vom Master repliziert:

```
$ mongo --port 9001 --eval db.buecher.find().forEach(printjson)
MongoDB shell version: 2.x
connecting to: 127.0.0.1:9001/test
```

Schreiboperationen auf den Slaves sind nicht erlaubt:

```
$ mongo --port 9000
MongoDB shell version: 2.x
connecting to: 127.0.0.1:9000/test
> db.buecher.insert({})
not master
```

```
$ mongo --port 9001
MongoDB shell version: 2.x
connecting to: 127.0.0.1:9001/test
> db.buecher.insert({})
not master
```

Auch der Slave verwaltet Meta-Informationen in einer Collection in der Datenbank `local`:

```
> use local
switched to db local
> show collections
me
sources
```

```
startup_log
system.indexes
> db.me.find()
{ "_id" : ObjectId("526a3cc2014449f1fdc068ab"), "host" : "<hostname>" }
> db.sources.find()
{ "host" : "localhost:27017", "source" : "main", "only" : "geheim" }
```

In der Collection `me` stehen (wenig interessante) Informationen über den Slave selbst, in der Collection `sources` (man beachte den Plural!) ist vermerkt, mit welchem Master die Replikation stattfindet. Anstatt der Kommandozeilenoption `--source` kann man auch ein Dokument in ebendiese Collection einfügen oder ein bestehendes ändern. So lässt sich im laufenden Betrieb der Master ändern.

Eine kaskadierende Replikation, bei der ein Slave seine Operationen wieder an einen anderen Slave weitergibt, ist nicht möglich, da auf einem Slave kein Oplog geführt wird.

Durch den Einsatz eines Master-Slave-Clusters stehen Ihnen folgende Möglichkeiten zur Verfügung:

- Sie können Lesezugriffe horizontal skalieren, da Sie sowohl vom Master als auch von allen Slaves lesen können. Dabei ist ggf. mit Inkonsistenzen zu rechnen für den Zeitraum, den die Replikation benötigt, um die auf dem Master geschriebenen Daten an die Slaves zu propagieren.
- Ausfallsicherheit für Lesezugriffe: Fällt der Master oder einer der Slaves aus, kann die Anwendung immer noch von den verbleibenden Knoten lesen.
- Reporting auf dedizierten Slaves: Lang laufende Auswertungen mit MapReduce oder dem Aggregation-Framework können auf einem der Slaves gefahren werden, ohne Ressourcen auf dem Master zu verbrauchen.

Die Nachteile eines Master-Slave-Betriebs sind genereller Natur. Der Master stellt einen Single Point of Failure dar. Fällt der Master aus, kann das Gesamtsystem nur noch eingeschränkt oder gar nicht arbeiten. Die statische Master-Slave-Architektur sollten Sie in produktiven Systemen besser nicht einsetzen. Verwenden Sie lieber ...

4.3 Replica Sets

Hinter dem Begriff *Replica Set* verbirgt sich eine Verallgemeinerung des Master-Slave-Konzepts, die die Ausfallsicherheit wesentlich erhöht. Dies wird durch eine flexiblere Rollenverteilung der einzelnen Knoten erreicht.

Zu jedem Zeitpunkt gibt es immer genau einen primären Knoten (genannt *Primary)* und mehrere sogenannte *Secondaries*. Der Primary entspricht dem Masterknoten; nur er führt schreibende Operationen innerhalb des Replica Set aus, die dann auf die Secondaries repliziert werden (siehe Abb. 4–2).