orangematter (https://orangematter.solarwinds.com)

An Introduction to MongoDB Replication and Replica Sets

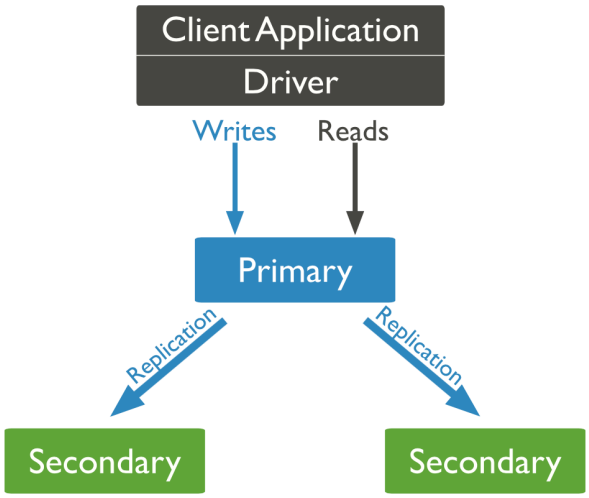
By SolarWinds (https://orangematter.solarwinds.com/author/solarwinds/)
August 25, 2017 | Database (https://orangematter.solarwinds.com/category/database/)

Replication is a process common to virtually all modern-day database systems. As you very likely know, it can be a complex su especially when dealing with nuances that vary from database type to database type. In this post we'd like to offer an introduct replication for users of MongoDB. Replication refers to a system's ability to automatically copy the data written to its database is stored in a secondary instance (or set of secondary instances). In other words, it's a mechanism that allows you to keep cop data in redundancy, and these can then be used for vital purposes, such as ensuring a backup that can maintain your system's availability, or recovering your data if your primary instance fails. This is true of replication in every kind of database. In Mongo however, there is a handful of specific characteristics you should know to make sure your replication runs as smoothly as poss

What Does Replication Give You in MongoDB?

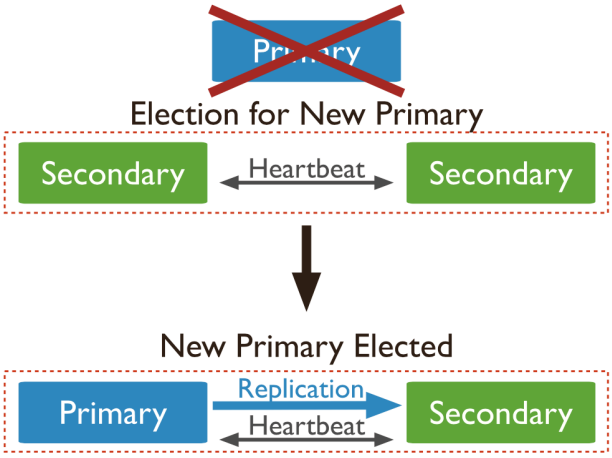
In general, MongoDB replication is more user-friendly and sophisticated than replication in traditional relational databases. For unlike some databases such as MySQL, MongoDB includes automatic failover (https://en.wikipedia.org/wiki/Failover), making inherently valuable for even beginner users who might not know how to set up failover manually. For comparison, MySQL users build and implement their failover systems themselves, or turn to third-party tools; MySQL doesn't provide a native option like MongoDB's. Overall, MongoDB's replication follows this trend, and it's relatively easy to set up, with many typically difficult aspe

already done by default.



Screenshot from MongoDB Manual 3.4 Documentatic

(https://docs.mongodb.com/manual/replication/) Similarly, adding a new replica set in MongoDB is easy, though there are a few things y to watch out for — the devil is in the details. If you don't plan ahead and know what to watch for, you may find yourself trapped handful of specific scenarios where it will become significantly more difficult to add new replicas. In these cases, you'll have to completely take down your replica set to fix the problem — an arduous process. These situations mostly arise due to complica the MongoDB oplog, which we'll cover in more depth later in this article. While MongoDB offers standard "primary-secondary" r it's much more common to use MongoDB's "replica sets." (https://docs.mongodb.com/manual/faq/replica-sets/) Replica sets multiple coordinated MongoDB instances, which work together to ensure superior availability. These are structured so that one is designated as the primary instance while the rest act as replica nodes. If the primary ever fails or becomes unavailable, one replicas will automatically be "elected" as the replacement. The election process is sophisticated and is a built-in element of M you can read more about the "Factors and Conditions that Affect Elections" here (https://docs.mongodb.com/manual/core/rep



elections/).

Screenshot from MongoDB Manual 3.4 Documentation

(https://docs.mongodb.com/manual/replication/) Getting started with MongoDB's replica sets is generally a user-friendly process — you to run a couple commands to set a replica up and even begin adding replicas. After this initial setup, instances will sync autom though users should also be aware that the system's initial sync can be time-consuming if there's a high volume of data.

Complications

Despite the general ease-of-use of MongoDB's replica sets, there's risk of complication when it comes to replica maintenance. biggest potential pitfalls are associated with MongoDB's "oplog." The oplog is a capped record of recent operations performed system, saved in the log to facilitate the repetition of any of those operations in the future; replicas sync via the oplog. As a "ca collection," the oplog has a maximum size — a size you, the user, designate during the startup of your MongoDB instance, usin oplogSizeMB (https://docs.mongodb.com/manual/reference/configuration-options/#replication.oplogSizeMB) option. (For My users familiar with managing the InnoDB transaction log, managing the oplog is similar in these regards.) Every time you make to the dataset, MongoDB writes a version of the change to the oplog. As the list of commands grows, if the oplog reaches cap automatically drop old records to make room for new ones. Risks with the oplog are related to its being capped: if your oplog i

small, any maintenance operations that require you to take down a replica can cause the copy to fail. If you find yourself in suc situation, you'll need to take down the primary node completely and allocate a larger oplog size to let it correctly sync. Furtherr amount of time your oplog permits you to have a node down is defined by your "oplog window" — the time difference between and last operation in the oplog. If adding a node to a replication set takes *longer to sync than the oplog window*, then the sync you can't sync a node fast enough, you need to, again, take down the primary and resize it — not an ideal situation.

Managing the Oplog Correctly

Tto avoid these fairly painful situations, you should expand your oplog as your dataset grows — we believe this is the ***one*** critic maintenance you must do for your MongoDB replica set. The correct size of oplog depends on two things:

- How often you modify the data in your database. The more data you're changing, the more operations you involve, and, the more you're logging to the oplog.
- The total amount of data you have.

As long as you keep these factors in mind, anticipate correct oplog size, and expand your oplog when necessary, you should b avoid any of MongoDB's most disruptive replication problems.

Another Maintenance Consideration

Beyond the oplog, another scaling and performance consideration is whether you should distribute your system's reads to nod your primary. By default, MongoDB clients will read from the primary in a replica set, but that characteristic can be controlled a adjusted per-query in applications, for scalability. You should be aware of this option and consider when it might be appropriat system to distribute reads. For reads and writes, the query level options for these adjustments are readConcern (https://docs.mongodb.com/manual/reference/read-concern/) and writeConcern (https://docs.mongodb.com/manual/referer concern/), which determine tolerance for consistency in your replica set. Using these options lets you determine the ways and which your queries read and write to specific replicas in your system. For example, with writeConcern, you can dictate how ma in your replica set a write must be applied to before the write is complete; this can range from requiring no acknowledgment at nodes in the replica set must have the write before we return." Distributing reads comes with a risk of consistency problems; ir you require consistent reads 100% of the time, then you should **always read from the primary.** Reads from secondaries can be problematic if you have extremely high standards for consistency rules. Reading from secondaries can also have advantages f scalability, but beware the trade-offs in consistency. And, if you need additional scalability but cannot tolerate stale reads in yo application, then sharding your data is probably the better solution (a topic we'll cover with more depth in a future article).

What MongoDB Replication Means for You

So, in summary, what are some of the bottom-line characteristics you can expect from MongoDB's replication capabilities?

- Your service will be highly available.
- You will have redundant copies of data, with the replication process automated and largely optimized by default.
- You will still need to consider your readConcern and writeConcern options.

You should also make an effort to pay special attention to these things:

- Replica lag — If secondaries get behind the primary, their reads will be inconsistent and potentially affect failovers.
- The oplog window — Remember, the "oplog window" is roughly the amount of time a node can be down before it requires sync. If the combined duration of downtime + the time it takes to sync a secondary takes longer than a primary's oplog wi secondaries cannot be added or synced until the primary itself is brought down and its oplog size is increased. Pay attent

Overall, MongoDB makes replication easy and effective, offering many of the most important things you should expect from re in a modern database technology. By making important processes — such as failover — accessible to even beginner users, Mc eases some of the pains associated with replication in other systems. Just remember the oplog!