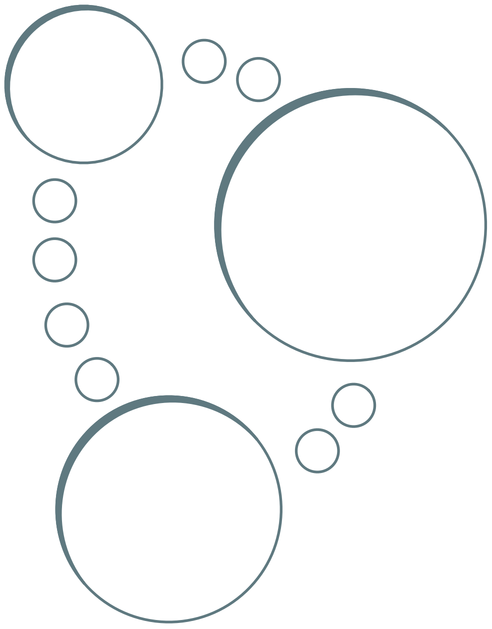
Understanding Neo4j Scalability

David Montag

January 2013



neotechnology
graphs are everywhere

Understanding Neo4j Scalability

Scalability means different things to different people. Common traits associated include:

1. Redundancy in the face of server failures, both for the data and for the operational service
2. Managing increasing read load
3. Managing increasing data set size
4. Managing increasing write load

The Neo4j scalability package is known as high availability, or HA. It primarily enables three things:

- Full data redundancy
- Service fault tolerance
- Linear read scalability

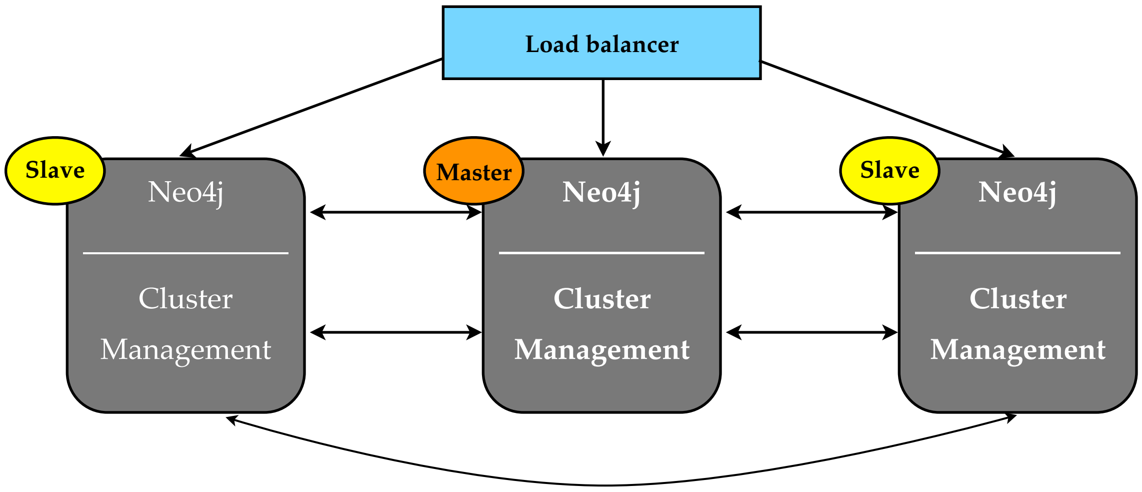
These features clearly address the first two scalability traits — redundancy and read throughput. We will first talk about exactly how Neo4j addresses these. Then we will discuss how to put HA to work addressing the remaining traits — managing growing datasets and increasing amounts of write load.

Lastly we will discuss a few common configurations of HA beyond the single main cluster, including:

- Online backups when the cluster is running
- Global cluster for data locality
- Disaster recovery for data center redundancy
- Reporting instances for ad-hoc reporting

Architecture

Neo4j high availability (HA) uses a master-slave cluster architecture, as shown here:



As the diagram illustrates, there are two parts to each Neo4j instance. One part is the database itself, and the other is the cluster management component. The cluster management component continuously stays in sync with all instances in the cluster, keeping track of any instances joining or leaving. When a master election becomes necessary, the cluster management component ensures that a new master is consistently elected. The database layer manages the rest of the system.

The Neo4j cluster performs automatic master election. Slave instances pull transactional updates of data from the master. As such, all write operations are coordinated by the master. It is still possible to write via slaves, but the slave will still make sure to perform the write operation synchronously with the master, behind the scenes. Therefore, when it comes to write performance, it is faster to write directly to the master than writing via a slave.

The astute reader will have realized that the write capacity of the cluster is constrained to that of the master. This is a correct observation, and relates to the fourth trait associated with scalability — managing increasing write load. This is something we will cover later in this paper.

Redundancy

In a Neo4j HA cluster, the full graph is replicated to each instance in the cluster. This means that the full dataset is replicated across the entire cluster, to each server. Regardless of the number of instances that fail, all the data is kept safe as long as one instance remains available. A consequence of this is that all data needs to fit within the capacity of a single Neo4j instance.

A single instance of Neo4j can house at most 34 billion nodes, 34 billion relationships, and 68 billion properties, in total. Businesses like Google obviously push these limits, but in general, this does not pose a limitation in practice. It is also important to understand that these limits were chosen purely as a storage optimization, and do not indicate any particular shortcoming of the product. They are easily, and are in fact being, increased.

Neo4j HA requires a quorum in order to serve write load. What this means is that a strict majority of the servers in the cluster need to be online in order for the cluster to accept write operations. For instance, three out of six servers will not do, it must be strictly more than half. This way, the Neo4j database service can continue to operate despite server outages. If enough servers are not available to form a quorum, the cluster will degrade into read-only operation until a quorum can be established.

Scaling Read Throughput

Although write operations are performed in unison with the elected master, read operations can be done locally on each slave. This means that the read capacity of the HA cluster increases linearly with the number of servers.

For example, if a three-instance cluster is operating at maximum capacity, serving 300 read requests per second, then adding a fourth instance would increase the capacity to 400 read requests per second.

Managing Large Datasets

A key trait of a graph database is the so-called index-free adjacency property. This means that a graph database can find the neighbors of any given node without having to consider the full set of relationships in the graph. A result of this is that, as dataset size increases, the time it takes to get the relationships off any given node stays constant. In contrast, the performance of a relational database is directly tied to the total number of rows in the table being queried. As the number of rows increases, performance degrades, regardless of index use.

Obviously this does not hold true for any kind of growth. At some point, Neo4j will hit resource constraints, such as limits in the amount of RAM available. At this point, performance will decrease when calculated across random requests, due to cache swapping. To address this, Neo4j makes use of a concept known as cache-based sharding.

Cache-based sharding is a very simple concept. The only thing it does is mandate consistent request routing. For instance, requests for user A are always sent to server 1, while requests for user B are always sent to server 2, and so on. The key assumption is that requests for user A typically touch parts of the graph around user A, such has his or her friends, preferences, likes, and so on. This means that the neighborhood of the graph around user A will be cached on server 1, while the neighborhood around user B will be cached on server 2. By employing consistent routing of requests, the caches of all servers in the HA cluster can be utilized maximally.

As outlined earlier, each server holds the full dataset. With cache-based sharding, each server caches a separate part of the graph, simply due to the way requests are routed. There will always be some overlap between the caches on different servers, but in general the strategy is highly effective for managing a large graph that does not fit in RAM.

Managing High Write Load

Neo4j HA makes use of a single master to coordinate all write operations, and is thus limited to the write throughput of a single machine. Despite this, write throughput can still be very high.

The first important thing to realize is that very few scenarios actually deal with sustained high write load. In the vast majority of cases, load is bursty with periods of high load and periods of silence or lower load. By introducing a queue for writes, a steady manageable stream of write operations can be serviced by the cluster. Queuing solutions have been widely successful in regulating load for backend systems. This can also introduce a number of desirable traits in a system, such as being able to batch write operations, thereby dramatically increasing performance, as well as having the ability to pause write operations without turning requests down during short periods of maintenance.

For exceptionally high write loads, the performance bottom line can easily be brought up by vertical scaling. Neo Technology actively supports customers operating on SSD drives such as Fusion-io, achieving exceptional write load and easily sustaining a rapidly growing business with plenty of capacity to spare.

Sharding Graphs

Today, many popular databases offer so-called sharding solutions. These function by partitioning the data across a number of servers. Many people opt for key-value or document databases for these exact reasons. A key thing to understand though, is that these solutions only store individual records. They are fundamentally unable to natively represent and query connections between records, and thus do not support things like referential integrity. Graph databases, spearheaded by Neo4j, are becoming the industry norm for the storage of any connected data.

The mathematical problem of optimally partitioning a graph across a set of servers is near-impossible (NP complete) to do for large graphs. Thus, this is a very hard problem to solve in a good way. Neo Technology is working on an offering of a partitioned flavor of Neo4j, currently set for release early 2014, subject to change of course. Customers looking to scale their business into the ultra-large graph space over the next few years can rest assured that Neo Technology will be accommodating them down the road.

Common HA Configurations

In enterprise settings, there are usually operational needs beyond what a single cluster can offer. All of the following strategies can be combined freely.

Online backups when the cluster is running

Today there is not a single successful business that operates without backups of their mission-critical data. Neo4j supports both full and incremental backups from running clusters.

Global cluster for data locality

For most online businesses, providing a fast and reliable service is paramount. Latency needs to be low, regardless of where a request is served. In order to serve a global audience, Neo4j can be configured to run as a global cluster. It operates by extending the main cluster with slave-only satellite clusters, typically operating read-only as well. This means that the master role still exists only in the main cluster, while satellite servers can be placed close to end-users. The satellite clusters stay up to date with the main cluster in real time.

Disaster recovery for data center redundancy

Mission-critical applications always make use of HA for data and service redundancy. A single cluster is however not protected in the adverse event of an entire data center failure, such as during a fire or natural disaster. In order to mitigate this risk, businesses with critical applications set up Neo4j with disaster recovery. Neo4j ensures that the full graph is replicated to a mirrored cluster in another data center. At the flip of a switch, the standby cluster switches into normal operations, and business can continue as usual.

Reporting instances for ad-hoc reporting

The need for ad-hoc reporting has been commonplace in the database world for decades, and Neo4j users expect nothing less. By setting up one or more reporting instances on the side of the main cluster, ad-hoc reporting and analytics jobs can be run without compromising production capacity. Most businesses at first do not realize what great data insights come with the use of a graph database like Neo4j. There is immense value in being able to draw insights from production data in real time. Reporting instances stay up to date with the main cluster like any other slave, but they will never be elected master.

Conclusion

Neo4j scales gracefully, both vertically and horizontally. What this means is that Neo4j can be run with 2GB or 200GB of RAM, regardless of dataset size. With cache-based sharding, graceful horizontal scaling is achieved across large datasets. Servers can be added until performance is satisfactory, and read performance scales linearly.

Looking back at the list of four scalability traits, we have covered how Neo4j addresses each and every one. HA brings dataset and service redundancy, as well as linear scaling of read capacity. Additionally, HA manages very large graphs by making use of cache-based sharding. In the case of high write load, it can typically be managed as an architectural concern by making use of standard technologies such as queues and fast hardware.

Lastly, businesses take advantage of common HA configurations to ensure their success, whether it comes to keeping latency low, keeping their data and service safe in the event of disaster, or drawing on real-time insights from the graph.