Must see: 5 Steps to Scaling MongoDB (Or Any DB) in 8 Minutes

Tuesday, September 13, 2011 at 9:07AM

General Chicken in Strategy, nosql

Jared Rosoff concisely, effectively, entertainingly, and convincingly gives an 8 minute MongoDB tutorial on scaling MongoDB at Scale Out Camp. The ideas aren't just limited to MongoDB, they work for most any database: Optimize your queries; Know your working set size; Tune your file system; Choose the right disks; Shard. Here's an explanation of all 5 strategies:

1. **Optimize your queries**. Computer science works. Complexity analysis works. A btree search is faster than a table scan. So analyze your queries. Use explain to see what your query is doing. If it is saying it's using a cursor then it's doing a table scan. That's slow. Look at the number of documents it looks at to satisfy a query. Look at how long it takes. Fix: add indexes. It doesn't matter if you are running on 1 or 100 servers.
2. **Know your working set size**. Sticking memcache in front of your database is silly. You have lots of RAM, use it. Embed your cache in the database, which is how MongoDB works. Working set = Active Documents + Used Indexes. Hitting something in RAM is fast, disk is slow. If you have a billion users and only 100K are active at a time then 100K is your working set. You want to have enough RAM for those 100K so operations are in RAM on not on disk. Remember indexes take memory too. It doesn't matter if you are running on 1 or 100 servers.

3. **Tune your file system**. Performance problems often traced to the filesystem. EXT3 is ancient. Use EXT4, XFS, or some other well performing file system. Turn off access time tracking, for a database there's no need to update a file every time a file is accessed, this is another write. Preallocating 2GB of files on EXT3 must actually write those bytes, it's slow.
4. **Choose the right disks**. Seek time is what matters. Most of what you are doing is random IO. Seek time is governed by a mechanical arm that has to swing over the disk. The average disk drive can do 200 seeks a second. Faster drives will move data off the disk faster, that is they have higher bandwidth, but their seek times will be the same. Single disk: you can do 200 queries a second. RAID 0 (stripe across multiple disks): 3 disks means 600 queries a second. RAID 10 (mirror and stripe): 6 disks means 1200 seeks a second. Choose the right disks. RAID matters. SSDs are awesome: .1 ms for a seek vs 5 ms for a disk seek. Great for random access.
5. **Shard**. If your app is slow, uses bad indexing, has slow disk drives, then a single node will be slow. Fix all this stuff before scaling out using sharded. Sharding lets you spread your workload over more machines along with high availability with replica sets. Data is partitioned to shards by ranges, for example. Can scale out to 100s of servers. Each can process 10s of 1000s of writes. Can add more capacity easily. Sharding with a good database multiples the benefits of having good queries, good drives, and good working sets.

Related Articles

- Hilarious Video: Relational Database Vs NoSQL Fanbois

Article originally appeared on High Scalability (http://highscalability.com/).