**Mongodb亿级数据量的性能测试**

笔者对三十五个非主流开源数据库中的佼佼者MongoDB进行了一下亿级数据量的性能测试，分别测试如下文中的几个项目。

AD：

进行了一下Mongodb亿级数据量的性能测试，分别测试如下几个项目：

（所有插入都是单线程进行，所有读取都是多线程进行）

1） 普通插入性能 （插入的数据每条大约在1KB左右）

2） 批量插入性能 （使用的是官方C#客户端的InsertBatch），这个测的是批量插入性能能有多少提高

3） 安全插入功能 （确保插入成功，使用的是SafeMode.True开关），这个测的是安全插入性能会差多少

4） 查询一个索引后的数字列，返回10条记录（也就是10KB）的性能，这个测的是索引查询的性能

5） 查询两个索引后的数字列，返回10条记录（每条记录只返回20字节左右的2个小字段）的性能，这个测的是返回小数据量以及多一个查询条件对性能的影响

6） 查询一个索引后的数字列，按照另一个索引的日期字段排序（索引建立的时候是倒序，排序也是倒序），并且Skip100条记录后返回10条记录的性能，这个测的是Skip和Order对性能的影响

7） 查询100条记录（也就是100KB）的性能（没有排序，没有条件），这个测的是大数据量的查询结果对性能的影响

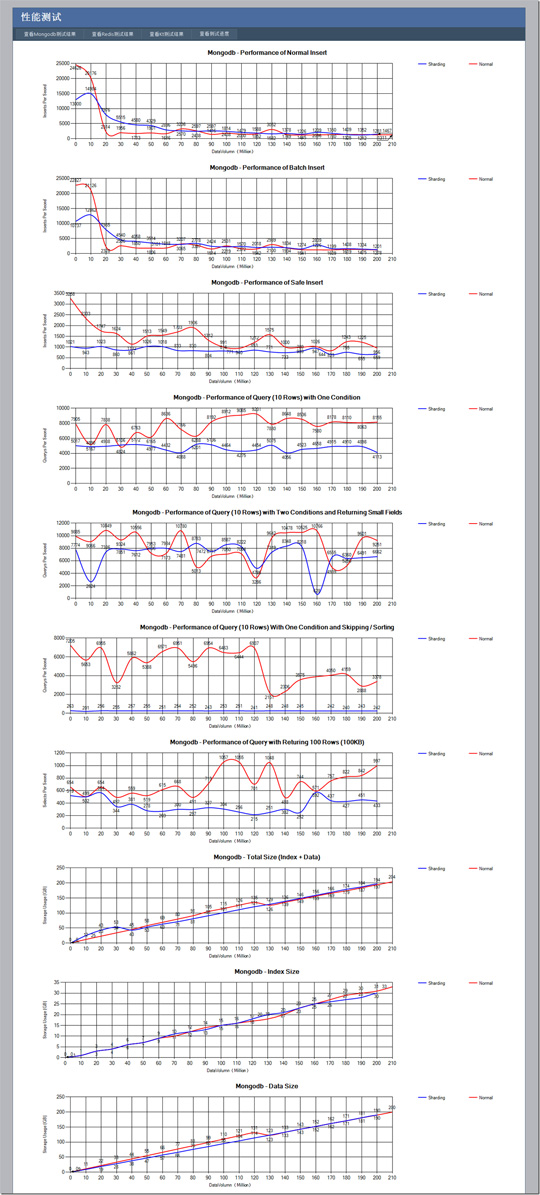
8） 统计随着测试的进行，总磁盘占用，索引磁盘占用以及数据磁盘占用的数量

并且每一种测试都使用单进程的Mongodb和同一台服务器开三个Mongodb进程作为Sharding（每一个进程大概只能用7GB左右的内存）两种方案

其实对于Sharding，虽然是一台机器放3个进程，但是在查询的时候每一个并行进程查询部分数据，再有运行于另外一个机器的mongos来汇总数据，理论上来说在某些情况下性能会有点提高

基于以上的种种假设，猜测某些情况性能会下降，某些情况性能会提高，那么来看一下最后的测试结果怎么样？

备注：测试的存储服务器是 E5620 @ 2.40GHz，24GB内存，CentOs操作系统，打压机器是E5504 @ 2.0GHz，4GB内存，Windows Server 2003操作系统，两者千兆网卡直连。

[](http://images.51cto.com/files/uploadimg/20110303/1045110.jpg)

从这个测试可以看出，对于单进程的方式：

1） Mongodb的非安全插入方式，在一开始插入性能是非常高的，但是在达到了两千万条数据之后性能骤减，这个时候恰巧是服务器24G内存基本占满的时候（随着测试的进行mongodb不断占据内存，一直到操作系统的内存全部占满），也就是说Mongodb的内存映射方式，使得数据全部在内存中的时候速度飞快，当部分数据需要换出到磁盘上之后，性能下降很厉害。（这个性能其实也不算太差，因为我们对三个列的数据做了索引，即使在内存满了之后每秒也能插入2MB的数据，在一开始更是每秒插入25MB数据）

2） 对于批量插入功能，其实是一次提交一批数据，但是相比一次一条插入性能并没有提高多少，一来是因为网络带宽已经成为了瓶颈，二来我想写锁也会是一个原因。

3） 对于安全插入功能，相对来说比较稳定，不会波动很大，我想可能是因为安全插入是确保数据直接持久化到磁盘的，而不是插入内存就完事。

4） 对于一列条件的查询，性能一直比较稳定，别小看，每秒能有8000-9000的查询次数，每次返回10KB，相当于每秒查询80MB数据，而且数据库记录是2亿之后还能维持这个水平，性能惊人。

5） 对于二列条件返回小数据的查询，总体上性能会比4）好一点，可能返回的数据量小对性能提高比较大，但是相对来说性能波动也厉害一点，可能多了一个条件就多了一个从磁盘换页的机会。

6） 对于一列数据外加Sort和Skip的查询，在数据量大了之后性能明显就变差了（此时是索引数据量超过内存大小的时候，不知道是否有联系），我猜想是Skip比较消耗性能，不过和4）相比性能也不是差距特别大。

7） 对于返回大数据的查询，一秒瓶颈也有800次左右，也就是80M数据，这就进一步说明了在有索引的情况下，顺序查询和按条件搜索性能是相差无几的，这个时候是IO和网络的瓶颈。

8） 在整个过程中索引占的数据量已经占到了总数据量的相当大比例，在达到1亿4千万数据量的时候，光索引就可以占据整个内存，此时查询性能还是非常高，插入性能也不算太差，mongodb的性能确实很牛。

那么在来看看Sharding模式有什么亮点：

1） 非安全插入和单进程的配置一样，在内存满了之后性能急剧下降。安全插入性能和单进程相比慢不少，但是非常稳定。

2） 对于一个条件和两个条件的查询，性能都比较稳定，但条件查询性能相当于单进程的一半，但是在多条件下有的时候甚至会比单进程高一点。我想这可能是某些时候数据块位于两个Sharding，这样Mongos会并行在两个Sharding查询，然后在把数据进行合并汇总，由于查询返回的数据量小，网络不太可能成为瓶颈了，使得Sharding才有出头的机会。

3） 对于Order和Skip的查询，Sharding方式的差距就出来了，我想主要性能损失可能在Order，因为我们并没有按照排序字段作为Sharding的Key，使用的是\_id作为Key，这样排序就比较难进行。

4） 对于返回大数据量的查询，Sharding方式其实和单进程差距不是很大，我想数据的转发可能是一个性能损耗的原因（虽然mongos位于打压机本机，但是数据始终是转手了一次）。

5） 对于磁盘空间的占用，两者其实是差不多的，其中的一些差距可能是因为多个进程都会多分配一点空间，加起来有的时候会比单进程多占用点磁盘（而那些占用比单进程少的地方其实是开始的编码错误，把实际数据大小和磁盘文件占用大小搞错了）。

虽然在最后由于时间的关系，没有测到10亿级别的数据量，但是通过这些数据已经可以证明Mongodb的性能是多么强劲了。另外一个原因是，在很多时候可能数据只达到千万我们就会对库进行拆分，不会让一个库的索引非常庞大。在测试的过程中还发现几个问题需要值得注意：

1） 在数据量很大的情况下，对服务进行重启，那么服务启动的初始化阶段，虽然可以接受数据的查询和修改，但是此时性能很差，因为mongodb会不断把数据从磁盘换入内存，此时的IO压力非常大。

2） 在数据量很大的情况下，如果服务没有正常关闭，那么Mongodb启动修复数据库的时间非常可观，在1.8中退出的-dur貌似可以解决这个问题，我简单测试了一下，开启dur对插入和查询性能影响都不是很大。

3） 在使用Sharding的时候，Mongodb时不时会对数据拆分搬迁，这个时候性能下降很厉害，虽然从测试图中看不出（因为我每一次测试都会测试比较多的迭代次数），但是我在实际观察中可以发现，在搬迁数据的时候每秒插入性能可能会低到几百条。

4） 对于数据的插入，如果使用多线程并不会带来性能的提高，反而还会下降一点性能（并且可以在http接口上看到，有大量的线程处于等待）。

5） 在整个测试过程中，批量插入的时候遇到过几次连接被远程计算机关闭的错误，怀疑是有的时候Mongodb不稳定关闭了连接，或是官方的C#客户端有BUG，但是也仅仅是在数据量特别大的时候遇到几次。