链滴

内存数据库FastDB和SQLite性能测评

作者：jetlan

原文链接：https://ld246.com/article/1376272100633

来源网站：链滴

  一、引言
  在很多项目中，经常会碰到这样的需求，需要对大量数据进行快速存储、查询、删除等操作，别是在一些针对诸如运营商、银行等大型企业的应用中，这些 需求尤为常见。比如智能网中的大量在并发用户的数据管理、软交换平台中的在线信息交互、宽带/3G等数据网中在线用户行为记录等等。
  针对这些情形，我们通常需要选择高性能的数据库产品，而且通常需要使用内存数据库，顾名义，内存数据库指的是所有的数据访问控制都在内存中进行， 这是与磁盘数据库相对而言的，磁盘数库虽然也有一定的缓存机制，但都不能避免从外设到内存的交换，而这种交换过程对性能的损耗是致的，目前主流数据库 如SYBASE、ORACLE等都有这种缓存机制，如将特定表绑定一定的缓存，从而一定程度上改善数据吞吐性能。而内存数据库几乎可以完全避免这种内外 存数据交换的发生，特别是物理内存足够大的设备上尤其如此，通常这种数据库也被称为主存数据库(Main Memory DataBase, MDB)。
  二、主存数据库比较
  目前比较知名的商业内存数据库有，ORACLE的TimesTen，MCObject的eXtremeDB、韩国的ltibase等，这些数据库 产品性能都非常的强劲，当然价格也相当的强劲，在非特大型系统建设时，常让人望而却步。于是退而求其次，免费开源内存数据库给了我们第二种选择。 Berkeley DB，SQLit，MonetDB，FastDB，H2等，不一而足。本文主要针对SQLite和FastDB进行性能测评。
  2.1 测试准备
  首先，笔者通过对评测数据的调研发现，通常认为，BDB性能不如SQLite，参考"免费的实时数据库，们该选谁？---BerkeleyDB与SQLite评测对比 "。
  上文中还提到，"据说FastDB很快，但数据库大小不能大于物理内存..."，于是笔对FastDB产生了兴趣，从FastDB作者的网站看到关于 这点的介绍，并不是说数据库大小不能大于物内存，而是说数据库大小超过物理内存时，性能与不超过时相比会有一定的降低(降低幅度未作说明，计是不推荐 使用)。幸运地是，目前物理内存实在说不上贵，服务器内存在10G之上都是很正常的事了。因此可以根据具体项目数据量需求来确定是否能使用 FastDB，比如并不是所有的表都需要放在存中。下面即将描述的测试表明，一旦使用FastDB，其性能在免费MMDB产品中绝对可执牛耳。由已经 有人对BDB和SQLite进行过比较，因此下面仅将FastDB与其中的优胜者SQLite进行性能测评。SLite采用内存模式，即打开数据库使使 用":memory:"参数，此时SQLite不产生数据库文，所有操作都在内存中，这一点需要特殊说明，与之不同的是，FastDB有两种模式，磁盘 模式和无模式，前者会产生磁盘文件，后者则与SQLite的内存模式相同。

 说是测评，其实过程也很简单，无非是设计测试CASE，编写测试CODE，输出测试RESULT，最做出结论。通常我们认为带索引的插入耗时相对 于查询和删除来说比较长，因此首先来看插入性能。用一个简单的表来完成接下来的所有测试，表中仅包含两个字段，INTEGER intKey，和VARCHAR stKey。测试平台为Window7 32bit系统(Evaluation Copy 7127)，编译器VC6 SP6。在DELL INSPIRN 640m上运行，CPU为Intel Core 2 CPU T5500 @ 1.66GHZ，内存2.5G。
 对FastDB（采用磁盘模式），表结构的定义如下：
 <blockquote>
  class _TestTable
{
public:
   b_int8 intKey;
    char const* strKey;
 &nsp; TYPE_DESCRIPTOR((KEY(intKey, INDEXED), KEY(strKey, INDEXED)));
};
 </blockquote>
 REGISTER(_TestTable);
 对SQLite，建表SQL如下：
 CREATE TABLE [_TestTable] ([intKey] INTEGER NOT NULL PRIMARY KEY, [strKey] ARCHAR(50) NULL)
 2.2 不同事务模式下的插入性能比较
 2.2.1 FastDB磁盘模式

我们首先按照批量事务处理的模式将intKey从1到nRecords(记录条数)，并指定相应的strKey，别调用相应的接口(均为原始 API)插入到两张表中，这里的批量事务处理模式指的是，比如插入1000条记录，插第一条之前开始事务，最后一条之后结束事务。此时在插入不同数目记 录时的表现分别如(一万条、十万条、72万条、一百万条)：
批量事务提交：
E:\intrest\FastDB\PerfTest\Debug>PerfTest.exe
[FASTDB] Elapsed time fr inserting 10000 record: 63 ms
[SQLITE] Elapsed time for inserting 10000 record: 639 ms
E:\intrest\FastDB\PerfTest\Debug>del *.fdb (清除测试生成数据，重新测试，下同。)

E:\intrest\FastDB\PerfTest\Debug>PerfTest.exe
[FASTDB] Elapsed time fr inserting 100000 record: 1186 ms
[SQLITE] Elapsed time for inserting 100000 rcord: 6318 ms
E:\intrest\FastDB\PerfTest\Debug>del *.fdb
E:\intrest\FastDB\PerfTest\Debug>PerfTest.exe
[FASTDB] Elapsed time fr inserting 7200000 record: 152460 ms
[SQLITE] Elapsed time for inserting 720000 record: 560121 ms
E:\intrest\FastDB\PerfTest\Debug>PerfTest.exe
[FASTDB] Elapsed time fr inserting 1000000 record: 15522 ms
[SQLITE] Elapsed time for inserting 100000 record: 67423 ms
从上我们可以看出，在批量事务模式下，FastDB比SQLite的插入性能提高了3-10倍。但是在很情况下，我们可能会需要逐条逐条的事务提交，下面给出了逐条事务模式的测试结果：
E:\intrest\FastDB\PerfTest\Debug>PerfTest.exe
[FASTDB] Elapsed time fr inserting 10000 record: 57315 ms（这个太恐怖了，不调整的话没法使用）
[SQLITE Elapsed time for inserting 10000 record: 780 ms
E:\intrest\FastDB\PerfTest\Debug>PerfTest.exe （SQLITE显式分条事务）
[ASTDB] Elapsed time for inserting 10000 record: 59967 ms
[SQLITE] Elapsed time for inserting 10000 record: 1154 ms
从上我们可以看出，FastDB在这种情形下的性能急遽降低，降到一个几乎不能接收的水平。经对FastDB的源代码分析(开源的好处体现出来 了)，发现FastDB在每次事务提交时，都会将变更的数内容同步到磁盘文件中（这是因为我们采用了磁盘模式），因此造成性能的显著降低。
直观上看，解决FastDB的这个问题有两种办法，一是避免每次事务提交时同步到磁盘，因为在种应用中，这种同步操作并不需要实时进行，通常每隔 一段时间同步一次就可以了(比如1S、1Min、根据具体项目的可靠性需要)；二是使用前面提到的FastDB无盘(DISKLESS)模式。
我们首先来看第一种方案，通过SEARCH FastDB文档(文档和社区是FastDB的一个软肋)，我们现作者已经考虑到了这个问题，FastDB为数据库提供了precommit的接口，用 于完成除sync到磁盘件外的所有事物操作，如释放mutex资源等。同时提供了backup接口，用来完成内存数据到磁盘文的备份，甚至支持打开数 据库时同时指定定时备份到磁盘文件的间隔。这样一来，每次事务提交的效理论上会得到大大提高，并且通过定时备份机制可以保证数据的可靠性。我们来看使用 precommit行逐条事务提交时FastDB的表现：
E:\intrest\FastDB\PerfTest\Debug>PerfTest（使用precommit逐条提交事务）
[FASTDB] Elapsed time for inserting 10000 record: 62 ms
[SQLITE] Elapsed tim for inserting 10000 record: 1170 ms
E:\intrest\FastDB\PerfTest\Debug>PerfTest
[FASTDB] Elapsed time for inerting 100000 record: 1170 ms
[SQLITE] Elapsed time for inserting 100000 record 11747 ms
E:\intrest\FastDB\PerfTest\Debug>PerfTest
[FASTDB] Elapsed time for inerting 1000000 record: 8081 ms
[SQLITE] Elapsed time for inserting 1000000 record: 125768 ms
从上可以看出，在逐条事务模式下，通过使用precommit技术，FastDB性能比SQLite提高了10左右。当然也许有读者怀疑加了备份 机制之后的性能，确实笔者没有进行这项测试，但是，需要注意是，FastDB在数据库关闭时会强制sync到磁盘文件，但SQLite没有这种功能，同 时，在进行这项测时，两种数据库都没有定时备份机制，因此该比较是公平的。
2.2.2 FastDB无盘模式

再来看第二种方案，FastDB采用无盘(通过编译选项控制生成DISKLESS版本)模式，此时FastDB始化一段共享内存(shmat or mmap)，这个初始大小通常很大，并且运行期不能扩展（无盘模式的劣）。我们将初始共享内存设置为1G，得到的测试结果如下：
E:\intrest\FastDB\PerfTest\Debug>PerfTest.exe
[FASTDB] Elapsed time for inserting 100000 record: 624 ms (批量事务提交)
[SQLITE] Elapsed time for inserting 100000 record: 11544 ms
E:\intrest\FastDB\PerfTest\Debug>PerfTest.exe
[FASTDB] Elapsed time for inserting 100000 record: 7410 ms (逐条事务提交)
[SQLITE] Elapsed time for inserting 100000 record: 11560 ms
E:\intrest\FastDB\PerfTest\Debug>PerfTest.exe
[FASTDB] Elapsed time for inserting 1000000 record: 134660 ms
[SQLITE] Elapsed time for inserting 100000 record: 120167 ms
E:\intrest\FastDB\PerfTest\Debug>PerfTest.exe
[FASTDB] Elapsed time for inserting 250000 record: 23666 ms
[SQLITE] Elapsed time for inserting 250000 ecord: 29110 ms
从上我们可以看出，无盘模式在大数据量下的表现与SQLite相近，这一点不是很好理解，需要究DISKLESS的设计模式，理论上应该与 precommit模式性能相近。但是实践是检验真理的唯一标准我们可以看出，磁盘模式的precommit方式性能表现卓越，不管从横向还是纵向来 看。
2.3 查询性能比较
下面的比较都使用磁盘模式的precommit方式，再来看索引查询的性能表现，测试时都是先插十万条数据后，再分别对该十万条数据进行查询，需 要注意的是我们同时对FastDB是否增加HASH引的性能进行了横向测评，FastDB增加HASH索引很简单，通过修改TYPE- DESCRIPTOR来完成，上的class中改为TYPE_DESCRIPTOR((KEY(intKey, INDEXED), KEY(strKey, INDEXED)));即为intKey增了Hash索引。
E:\intrest\FastDB\PerfTest\Debug>perftest (FASTDB哈希索引)
[FASTDB] Elapsed time for inserting 100000 record: 624 ms
[FASTDB] Elapsed time for 10000 index searches: 328 ms
[SQLITE] Elapsed time for inserting 100000 record: 1032 ms
[SQLITE] Elapsed time for 100000 index searches: 10935 ms
E:\intrest\FastDB\PerfTest\Debug>perftest(FASTDB非哈希索引)
[FASTDB] Elapsed time for inserting 100000 record: 577 ms
[FASTDB] Elapsed time for 10000 index searches: 515 ms
[SQLITE] Elapsed time for inserting 100000 record: 1043 ms
[SQLITE] Elapsed time for 100000 index searches: 9532 ms
从测试结果可以看出，查询十万条索引记录的效率，FastDB要比SQLite快20倍左右，并且在增HASH索引后能够得到进一步的改善。
2.4 删除性能比较及综合表现
最后，我们在测试删除效率时，同时综合来看FastDB与SQLite之间插入、查询、删除的性能表：
插入、查询、删除综合比较：
E:\intrest\FastDB\PerfTest\Debug>perftest（批量删除，FASTDB.removeall()，SQLITE.delete*）
[FASTDB] Elapsed time for inserting 100000 record: 608 ms
FASTDB] Elapsed time for 100000 index searches: 687 ms
[FASTDB] Elapsed time for deleting all 100000 records: 16 ms
[SQLITE] Elapsed time for inserting 100000 record: 11107 ms
[SQLITE] Elapsed time for 100000 index searches: 10062 ms&bsp;
[SQLITE] Elapsed time for deleting all 100000 records: 16 ms
E:\intrest\FastDB\PerfTest\Debug>perftest（逐条删除）
[FASTDB] Elapse time for inserting 100000 record: 593 ms
[FASTDB] Elapsed time for 100000 index searches: 562 ms
[FASTDB] Elapsed time for deleting all 100000 records one by one: 905 ms
[SQLITE] Elapsed time for inserting 100000 record: 10406 ms
[SQLITE] Elapsed time for 100000 index searches: 10249 ms
[SQLITE] Elaped time for deleting all 100000 records one by one: 8923 ms
从上可以看出，就删除效率而言，批量删除的速度二者相近，而逐条删除时，十万条记录的删累积，FastDB比SQLite快了10倍左右。

2.5 总结

优点：FastDB磁盘模式下，采用precommit方式，性能远远优于SQLite，并且FastDB提供了善的备份恢复机制，能够保证数据 安全。FastDB的无盘模式在小数据量时表现优越，并且不会产生盘数据文件，也不能加载已经保存的数据库文件，看起来更像是针对嵌入式设备(如智能手 机、PDA等开发的，对于这种场景可以考虑使用无盘模式。
缺点：FastDB目前能够SEARCH到的比较著名的应用是PingTel公司的开源统一通信产品SIPX，产品采用的是FastDB的磁盘模 式。这可能多少与FastDB的完全授权模式有关，而SQLite采用的是GP的不允许闭源的商业发布。当然主要还是社区的不成熟，这从Google Trends的搜索结果也能看出。区的不成熟会带来学习成本的增加，这一点在选型时也需要考虑。