从MySQL到MongoDB

——视觉中国的NoSQL之路

■ 文/ 潘凡

起因

视觉中国网站(www.chinavisual.com)是国内最大的创意人群的专业网站。2009年以前，同很多公司一样，我们的CMS和社区产品都构建于PHP+Nginx+MySQL之上：MySQL使用了Master+Master的部署方案；前端使用自己的PHP框架进行开发；Memcached作为缓存；Nginx进行Web服务和负载均衡；Gearman进行异步任务处理。在传统的基于静态内容（如文章，资讯，帖子）的产品，这个体系运行良好。通过分级的缓存，数据库端实际负载很轻。2009年初，我们进行了新产品的开发。此时，我们遇到了如下一些问题。

用户数据激增：我们的MySQL某个信息表上线1个月的数据就达到千万。我们之前忽略的很多数据，在新形势下需要跟踪记录，这也导致了数据量的激增；

用户对于信息的实时性要求更高：对信息的响应速度和更新频度就要求更高。简单通过缓存解决的灵丹妙药不复存在；

对于Scale-out的要求更高：有些创新产品的增长速度是惊人的。因此要求能够无痛的升级扩展，否则一旦停机，那么用户流失的速度也是惊人的；

大量文件的备份工作：我们面向的是创意人群，产生的内容是以图片为主。需要能够对这些图片及不同尺寸的缩略图进行有效的备份管理。我们之前使用的Linux inotify+rsync的增量备份方案效果不佳；

需求变化频繁：开发要更加敏捷，开发成本和维护成本要更低，要能够快速地更新进化，新功能要在最短的周期内上线。

最初，我们试图完全通过优化现有的技术架构来解决以上问题：对数据时效性进一步分级分层缓存，减小缓存粒度；改进缓存更新机制（线上实时和线下异步更新）提高缓存命中率；尝试对业务数据的特点按照水平和垂直进行分表；使用MogileFS进行分布存储；进一步优化Mysql的性能，同时增加MySQL节点等。但很快发现，即便实施了上述方案，也很难完全解决存在的问题：过度依赖Memcached导致数据表面一致性的维护过于复杂，应用程序开发需要很小心，很多时候出现Memcached的失效会瞬间导致后端数据库压力过大；不同类型数据的特点不同，数据量差别也很大；分表的机制和方式在效率平衡上很难取舍；MogileFS对我们而言是脚小鞋大，维护成本远远超过了实际的效益；引入更多的MySQL数据库节点增大了我们的维护量，如何有效监控和管理这些节点又成了新的问题。虽然虚拟化可以解决部分问题，但还是不能令人满意；

除了MySQL，能否找到一个更为简单、轻便的瑞士军刀呢？我们的目光投向了NoSQL的方案。

候选方案

最初，对于NoSQL的候选方案，我依据关注和熟悉程度，并且在甄别和选择合适的方案时特别制定了一些原则：是否节省系统资源，对于CPU等资源是否消耗过大；客户端/API支持，这直接影响应用开发的效率；文档是否齐全，社区是否活跃；部署是否简单；未来扩展能力。按以上几点经过一段测试后，我们候选名单中剩下Redis、MongoDB和Flare。

Redis对丰富数据类型的操作很吸引人，可以轻松解决一些应用场景，其读写性能也相当高，唯一缺点就是存储能力和内存挂钩，这样如果存储大量的数据需要消耗太多的内存（最新的版本已经不存在这个问题）。

Flare的集群管理能力令人印象深刻，它可以支持节点的动态部署，支持节点的基于权重的负载均衡，支持数据分区。同时允许存储大的数据，其key的长度也不受Memcached的限制。而这些对于客户端是透明的，客户端使用Memcached协议链接到Flare的proxy节点就可以了。由于使用集群，Flare支持fail－over，当某个数据节点宕掉，对于这个节点的访问都会自动被proxy节点forward到对应的后备节点，恢复后还可以自动同步。Flare的缺点是实际应用案例较少，文档较为简单，目前只在Geek使用。

以上方案都打算作为一个优化方案，我从未想过完全放弃MySQL。然而，用MongoDB做产品的设计原型后，我彻底被征服了，决定全面从MySQL迁移到MongoDB。

为什么MongoDB可以替代MySQL？

MongoDB是一个面向文档的数据库，目前由10gen开发并维护，它的功能丰富，齐全，完全可以替代MySQL。在使用MongoDB做产品原型的过程中，我们总结了MonogDB的一些亮点：

使用JSON风格语法，易于掌握和理解：MongoDB使用JSON的变种BSON作为内部存储的格式和语法。针对MongoDB的操作都使用JSON风格语法，客户端提交或接收的数据都使用JSON形式来展现。相对于SQL来说，更加直

观，容易理解和掌握。

Schema-less，支持嵌入子文档：MongoDB是一个Schema-free的文档数据库。一个数据库可以有多个Collection，每个Collection是Documents的集合。Collection和Document和传统数据库的Table和Row并不对等。无需事先定义Collection，随时可以创建。

Collection中可以包含具有不同schema的文档记录。这意味着，你上一条记录中的文档有3个属性，而下一条记录的文档可以有10个属性，属性的类型既可以是基本的数据类型（如数字、字符串、日期等），也可以是数组或者散列，甚至还可以是一个子文档（embed document）。这样，可以实现逆规范化（denormalizing）的数据模型，提高查询的速度。

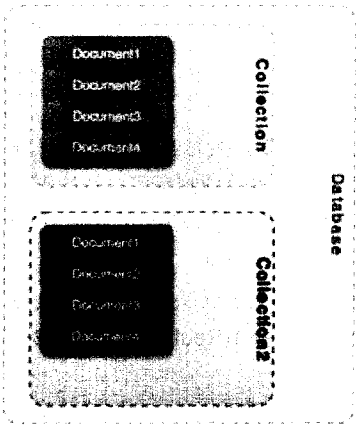


图1 MongoDB是一个Schema-free的文档数据库

图2是一个例子，作品和评论可以设计为一个collection，评论作为子文档内嵌在art的comments属性中，评论的回复则作为comment子文档的子文档内嵌于replies属性。按照这种设计模式，只需要按照作品id检索一次，即可获得所有相关的信息了。在MongoDB中，不强调一定对数据进行Normalize，很多场合都建议De-normalize，开发人员可以扔掉传统关系数据库各种范式的限制，不需要把所有的实体都映射为一个Collection，只需定义最顶级的class。MongoDB的文档模型可以让我们很轻松就能将自己的Object映射到collection中实现存储。

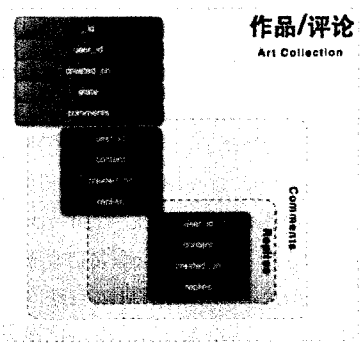


图2 MongoDB支持嵌入子文档

简单易用的查询方式：MongoDB中的查询让人很舒适，没有SQL难记的语法，直接使用JSON，相当的直观。对不同的开发语言，你可以使用它最基本的数组或散列格式进行查询。配合附加的operator，MongoDB支持范围查询，正则表达式查询，对子文档内属性的查询，可以取代原来大多数任务的SQL查询。

CRUD更加简单，支持in-place update：只要定义一个数组，然后传递给MongoDB的insert/update方法就可自动插入或更新；对于更新模式，MongoDB支持一个upsert选项，即："如果记录存在那么更新，否则插入"。MongoDB的update方法还支持Modifier，通过Modifier可实现在服务端即时更新，省去客户端和服务端的通讯。这些modifer可以让MongoDB具有和Redis、Memcached等KV类似的功能：较之MySQL，MonoDB更加简单快速。Modifier也是MongoDB可以作为对用户行为跟踪的容器。在实际中使用Modifier来将用户的交互行为快速保存到MongoDB中以便后期进行统计分析和个性化定制。

所有的属性类型都支持索引，甚至数组：这可以让某些任务实现起来非常的轻松。在MongoDB中，"_id"属性是主键，默认MongoDB会对_id创建一个唯一索引。

服务端脚本和Map/Reduce：MongoDB允许在服务端执行脚本，可以用Javascript编写某个函数，直接在服务端执行，也可以把函数的定义存储在服务端，下次直接调用即可。MongoDB不支持事务级别的锁定，对于某些需要自定义的"原子性"操作，可以使用Server side脚本来实现，此时整个MongoDB处于锁定状态。Map/Reduce也是MongoDB中比较吸引人的特性。Map/Reduce可以对大数据量的表进行统计、分类、合并的工作，完成原先SQL的GroupBy等聚合函数的功能。并且Mapper和Reducer的定义都是用Javascript来定义服务端脚本。

性能高效，速度快：MongoDB使用c++/boost编写，在多数场合，其查询速度对比MySQL要快的多，对于CPU占用非常小。部署也很简单，对大多数系统，只需下载后二进制包解压就可以直接运行，几乎是零配置。

支持多种复制模式：MongoDB支持不同的服务器间进行复制，包括双机互备的容错方案。

Master-Slave是最常见的。通过Master-Slave可以实现数据的备份。在我们的实践中，我们使用的是Master-Slave模式，Slave只用于后备，实际的读写都是从Master节点执行。

Replica Pairs/Replica Sets允许2个MongoDB相互监听，实现双机互备的容错。

MongoDB只能支持有限的双主模式（Master-Master），实际可用性不强，可忽略。

内置GridFS，支持大容量的存储：这个特点是最吸引我眼球的，也是让我放弃其他NoSQL的一个原因。GridFS具体实现其实很简单，本质仍然是将文件分块后存储到files.file和files.chunk 2个collection中，在各个主流的driver实现中，都封装了对于GridFS的操作。由于GridFS自身也是一个Collection，你可以直接对文件的属性进行定义和管理，通过这些属性就可以快速找到所需要的文件，轻松管理海量的文件，无需费神如何hash才能避免文件系统检索性能问

题，结合下面的Auto-sharding，GridFS的扩展能力是足够我们使用了。在实践中，我们用MongoDB的GridFs存储图片和各种尺寸的缩略图。

内置Sharding，提供基于Range的Auto Sharding机制：一个collection可按照记录的范围，分成若干个段，切分到不同的Shard上。Shards可以和复制结合，配合Replica sets能够实现Sharding+fail-over，不同的Shard之间可以负载均衡。查询是对客户端是透明的。客户端执行查询，统计，MapReduce等操作，这些会被MongoDB自动路由到后端的数据节点。这让我们关注于自己的业务，适当的时候可以无痛的升级。MongoDB的Sharding设计能力最大可支持约20 petabytes，足以支撑一般应用。

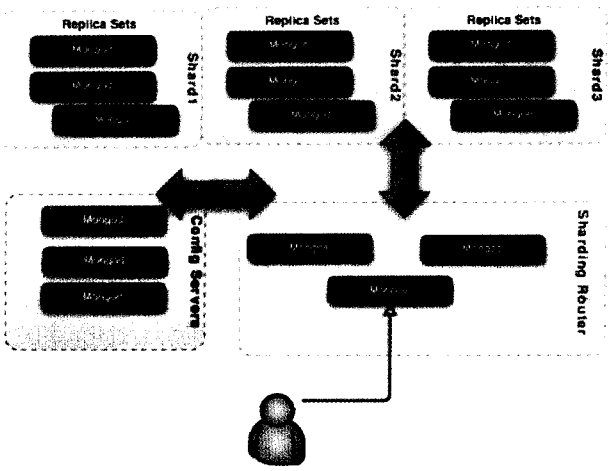


图3 MongoDB的Auto-sharding结构

第三方支持丰富：MongoDB社区非常活跃，很多开发框架都迅速提供了对MongDB的支持。不少知名大公司和网站也在生产环境中使用MongoDB，越来越多的创新型企业转而使用MongoDB作为和Django，RoR来搭配的技术方案。

实施结果

实施MonoDB的过程是令人愉快的。我们对自己的PHP开发框架进行了修改以适应MongoDB。在PHP中，对MongoDB的查询、更新都是围绕Array进行的，实现代码变得很简洁。由于无需建表，MonoDB运行测试单元所需要的时间大大缩短，对于TDD敏捷开发的效率也提高了。当然，由于MongoDB的文档模型和关系数据库有很大不同，在实践中也有很多的困惑，幸运的是，MongoDB开源社区给了我们很大帮助。最终，我们使用了2周就完成了从MySQL到MongoDB的代码移植比预期的开发时间大大缩短。从我们的测试结果看也是非常惊人，数据量约2千万，数据库300G的情况下，读写2000rps，CPU等系统消耗是相当的低（我们的数据量还偏小，目前陆续有些公司也展示了他们的经典案例：MongoDB存储的数据量已超过 50亿，>1.5TB）。目前，我们将MongoDB和其他服务共同部署在一起，大大节约了资源。

一些小提示

切实领会MongoDB的Document模型，从实际出发，扔掉关系数据库的范式思维定义，重新设计类；在服务端运行的JavaScript代码避免使用遍历记录这种耗时的操作，相反要用Map/Reduce来完成这种表数据的处理；属性的类型插入和查询时应该保持一致。若插入时是字符串“1”，则查询时用数字1是不匹配的；优化MongoDB的性能可以从磁盘速度和内存着手；MongoDB对每个Document的限制是最大不超过4MB；在符合上述条件下多启用Embed Document，避免使用DatabaseReference；内部缓存可以避免N+1次查询问题（MongoDB不支持joins）。

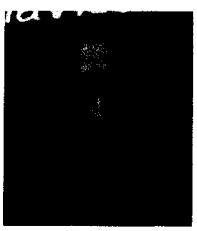
用Capped Collection解决需要高速写入的场合，如实时日志；大数据量情况下，新建同步时要调高oplogSize的大小，并且自己预先生成数据文件，避免出现客户端超时；Collection+Index合计数量默认不能超过24000；当前版本（<v1.6）删除数据的空间不能被回收，如果你频繁删除数据，那么需要定期执行repairDatabase，释放这些空间。

结束语

MongoDB的里程碑是1.6版本，预计今年7月份发布，届时，MongoDB的Sharding将首次具备在生产环境中使用的条件。作为MongoDB的受益者，我们目前也在积极参与MongoDB社区活动，改进Perl/PHP对于MongoDB的技术方案。在1.6版本后也将年内推出基于MongoDB的一些开源项目。

对于那些刚刚起步，或者正在开发创新型互联网应用的公司来说，MongoDB的快速、灵活、轻量和强大扩展性，正适合我们快速开发产品，快速迭代，适应用户迅速变化和更新的种种需求。

总而言之，MongoDB是一个最适合替代MySQL的全功能的NoSQL产品，使用MongoDB+Perl/PHP/Django/RoR的组合将很快成为开发Web2.0、3.0的产品的最佳组合，就像当年MySQL替代Oracle/DB2/Informix一样，历史总是惊人的相似，让我们拭目以待吧！



作者简介：

潘凡(nightsailer，N.S.)，视觉中国网站技术总监，联合创始人，家有1狗2猫。目前负责网站平台设计和底层产品研发工作。当前关注：Apps平台设计、分布式文件存储、NoSQL、高性能后现代的Perl编程。Twitter：@nightsailer Blog：http://nightsailer.com/