一种 MongoDB 应用优化策略①

卢至彤, 李 翀, 柯 勇, 孙健英

(中国科学院 计算机网络信息中心, 北京 100190)

摘 要: 为了解决不断增长的文件存储需求, 和高流量高并发的访问量, 增强系统的响应性能, 提出一种 Web 应用优化策略, 通过 MongoDB GridFS 对网站文件存储容量、可用性和可靠性进行扩展, 并且通过 Nginx 和 Keepalived, 对后台访问进行负载均衡和双机热备, 优化并发性能. 实验表明, 当并发访问数上升至 80 以上时, 平均访问响应时间缩短 9%. 文件通过 Nginx Gridfs 进行高并发上传时非常稳定, 在较大文件下载时比直接通过本地文件系统 EXT4 下载速度更高.

关键词: MongoDB; GridFS; Nginx; Keepalived; concurrency

Optimisation Strategy for Web Applications Based on MongoDB

LU Zhi-Tong, LI Zhong, KE Yong, SUN Jian-Ying

(Computer Network Information Center, Chinese Academy of Sciences, Beijing 100190, China)

Abstract: With the growing demand for massive file storage and high number of concurrent accesses to enhance performance of the system. In this paper, we propose an optimization strategy of web applications, which can expand the storage capacity, availability and reliability with MongoDB GridFS, and also can support load balancing and hot standby with Nginx and Keepalived, optimizing concurrent performance. We test the concurrent performances of the web application and the experimental results show that the average response time could be reduce by 9% when the number of concurrent accesses increases to more than 80. When files are uploaded concurrently through Nginx Gridfs, the performance is stable .The file-download speeds of larger files are faster than those through local file system EXT4.

Key words: MongoDB; GridFS; Nginx; Keepalived; concurrency

Web 应用通过文件服务器对图片、视频和文档资源对文件进行管理, 包括: 文件存储、文件同步和文件访问. 对于大多数应用而言, 需要文件系统来存储用户上传的文件. 一般而言, 用户上传的单个文件所占用的空间一般不大, 但是上传的文件数量是在不断快速增长的. 单机文件系统存储容量可能会超过单机硬盘的扩容范围而且其查询性能在存储量级过大时查找和插入性能都可能会遇到瓶颈. 我们可以采用 MongoDB GridFS[1]分布式文件系统来达到我们动态增加存储容量的目的.

除此以外, MongoDB GridFS 分布式文件系统能够自动进行冗余备份, 保证用户的文件不会丢失; 还有高可用性, 也就是说当某个文件服务器出现故障的时候, 自动切换到备份提供服务, 使用户感觉不到有什么异常.

随着 Web 应用访问量的提高, 我们可以通过 Nginx 反向代理服务器的负载均衡来提升 Web 应用的响应性能. 对于负载均衡集群架构系统, 各服务器间需要共享 session 信息, 我们可以通过 Memcache[2]这个高性能的分布式的内存对象缓存系统来解决这个问题.

本文通过 MongoDB GridFS 分布式文件系统对网站文件存储容量、可用性和读取性能进行扩展. 通过 Nginx[3]代理服务器大量的并发访问或数据流量分担到多台节点设备上分别处理, 减少用户等待响应的时间, 优化并发性能.

① 收稿时间:2016-09-05;收到修改稿时间:2016-10-17 [doi:10.15888/j.cnki.csa.005767]

1　系统设计

系统设计架构示意图如图 1 所示, 系统通过 Nginx 代理服务器实现对于业务服务器和文件服务器集群的访问, 其中系统使用两台代理服务器采用双机热备技术确保系统的可靠性. 通过设置负载均衡集群将访问业务分摊到两台业务服务器上, 降低了单个业务服务器的访问压力. 主从代理服务器的负载均衡是使用 Nginx 作为反向代理服务器来实现的.

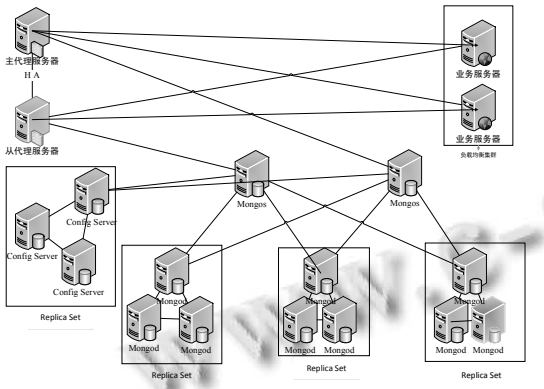

图 1　系统架构图

分布式文件系统 MongoDB GridFS 部署在 MongoDB 分布式数据库上. 数据库分成三个分片[4]进行横向扩展. 每个分片由 3 台 Mongod Server 副本集组成以保证可靠性.

1.1 负载均衡集群架构

系统使用 2 台 Nginx 代理服务器作为系统的出口, 通过一个虚拟 IP 对外提供服务, 如下图 2 所示. 为保证系统的高可用性, 采用双机热备模式. 两台代理服务器同一时间只有一台在提供服务. 当提供服务的一台出现故障的时候, 另外一台会马上自动接管并且提供服务, 进行无缝交接. 双机热备是通过路由冗余协议在 2 台代理服务器上分别安装 KeepAlive 并进行配置实现的.

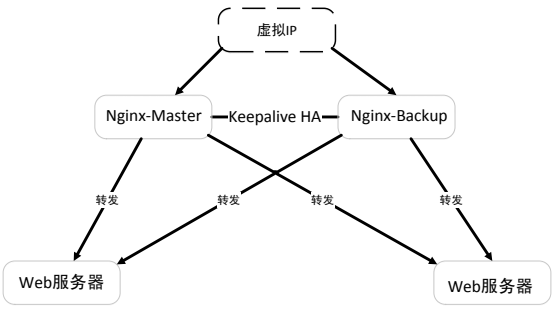

图 2　负载均衡集群架构

1.2 MongoDB 副本集

副本集架构如下图 3 所示, 系统中一共包含 4 个副本集, 每个副本集包含 3 个副本, 分别是一个 Primary, 2 个 Secondary. 3 个副本在同步之后存储的是同一份数据. 其中, 主节点负责整个副本集的数据写入, 从节点定期从主节点通过 Oplog 同步数据备份, 通过配置可从距离最近节点读取数据, 实现读写分离. 副本集之间通过心跳维持联系, 一但主节点挂掉失去联系, 从节点就会选举一个新的主节点, 选举过程对客户端是透明的. 副本集提供了数据的冗余备份, 并因在多个服务器上可读取存储的数据副本, 提高了数据的可用性和故障容忍性.

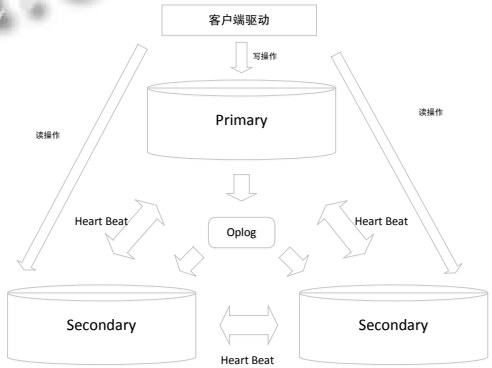

图 3　副本集架构图

1.3 MongoDB 分片架构

为提高存储空间, 分担请求负载, 采用 MongoDB 集群的分片配置[5], 如下图 4 所示. 整个分片集群不同服务器分别承担以下不同角色:

① Mongos Server, 路由服务器, 数据库集群请求的入口, 负责把对应的数据请求请求转发到对应的分片服务器上. 管理操作、读写操作都通过 mongos server 来完成, 以保证集群多个组件处于一致的状态.

② Config Server, 配置服务器, 存储所有数据库元信息的配置, 即各个 chunk 与分片服务器的映射关系. 如下图所示, 将配置服务器配置成一个副本集防止系统单点故障.

③ Shard Server, 分片服务器, 存储数据库中具体的的数据. 其中, 每一个分片服务器是一个副本集保证数据可用性. 如下图所示, 集群有 3 个分片服务器, 必要时还可进行扩展.

在分片服务器里, MongoDB 会把数据根据片键分为 chunks, 如图 5 所示. 当一个 chunk 的大小超过配置

中的 chunk size 时, MongoDB 的后台进程会把这个 chunk 切分成更小的 chunk. 除此之外, MongDB 的后台进程 Balancer 负责 chunk 的迁移, 从而均衡各个分片服务器的负载. 这些过程对客户端都是透明的.

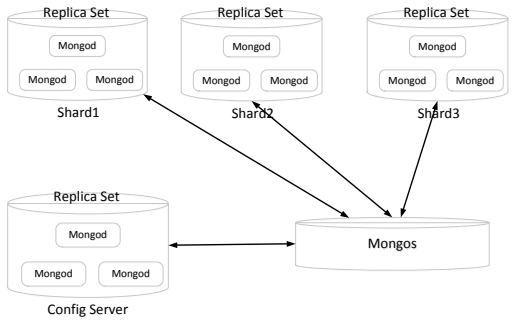

图 4 MongoDB 分片架构图

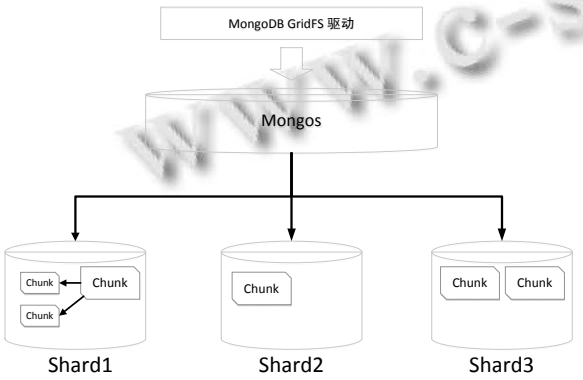

图 5 MongoDB chunk 分布图

1.4 GridFS

GridFS[6]是 MongoDB 之上的分布式文件系统, 通过 MongoDB 的复制, 分片等机制来存储文件数据和文件元数据并进行管理与分析. GridFS 将二进制数据大文件分成很多块, 每一块作为一个单独的文档存储.

GridFS 使用两个文档来存储二进制数据文件, 一个用来存储文件本身的块, 另外一个用来存储分块的信息和文件的元数据, 默认对应的集合分别为 fs.chunks 和 fs.files, 其结构如图 6 所示.

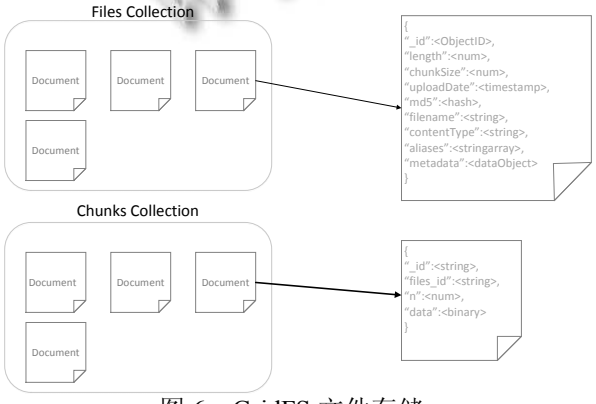

图 6 GridFS 文件存储

fs.files 这个集合 Collection 存储文件元数据, 一般比较小, 不需要分片存储. fs.chunks 这个集合 Collection 存储了文件数据, 比较大, 根据 files_id,n 作为片键将不同 chunk 通过哈希分布到不同分片服务器上.

2 系统实现

2.1 运行环境

运行环境集群中有 7 台服务器, 其环境如表 2 所示, 其中每台服务器的配置环境如表 1 所示.

表 1 服务器配置信息

属性	配置信息
CPU	4 核 Intel(R) Xeon(R) CPU E5649 @ 2.53GHz
Memory	8MB
Disk	38.7 GB
NetWork	BandWidth 100MBs
JVM Version	Java 1.7.0_71

2.2 配置

2.2.1 MongoDB 和 GridFS 配置

MongoDB 的配置[7]示意图如下图所示. 在如下的配置中, 任何一台机器因故障不能提供服务, 都能保证 MongoDB 继续提供正常的服务, 数据不丢失, 保证容灾性. 并且在相应的配置后, 能增加和减少节点, 有良好的可扩展性.

在四个服务器上开启 Mongod 服务, 然后对 Shard1、Shard2、Shard3 和 configReplSet 副本集用 mongo 命令进行连接, 然后在 Mongo Shell 中配置副本集信息, 如下所示.

```
>rs.initiate({"_id":"shard1","version":1,"members":[{"_id":0,"host":"10.10.1.134:27001","priority":10},{"_id":1,"host":"10.10.1.135:27001","priority":5},{"_id":2,"host":"10.10.1.136:27001","arbiterOnly":true}]})
>rs.initiate({"_id":"shard2","version":1,"members":[{"_id":0,"host":"10.10.1.135:27002","priority":10},{"_id":1,"host":"10.10.1.136:27002","priority":5},{"_id":2,"host":"10.10.1.137:27002","arbiterOnly":true}]})
>rs.initiate({"_id":"shard3","version":1,"members":[{"_id":0,"host":"10.10.1.137:27003","priority":10},{"_id":1,"host":"10.10.1.134:27003","priority":5},{"_id":2,"host":"10.10.1.136:27003","arbiterOnly":true}]})
```

>rs.initiate({"_id":"configReplSet","version":1,"members":[{"_id":0,"host":"10.10.1.137:27000","priority":10},{"_id":1,"host":"10.10.1.134:27000","priority":1},{"_id":2,"host":"10.10.1.135:27000","priority":1}]})

表 2　服务器集群环境

序号	服务器名称	IP 地址	环境	说明
1	服务器 1	10.10.1.134	CentOS release6.5 x86_64	MongoDB GridFS 文件服务器
2	服务器 2	10.10.1.135	CentOS release6.5 x86_64	MongoDB GridFS 文件服务器
3	服务器 3	10.10.1.136	CentOS release6.5 x86_64	MongoDB GridFS 文件服务器
4	服务器 4	10.10.1.137	CentOS release6.5x86_64	MongoDB GridFS 文件服务器
5	服务器 6	10.10.1.138	CentOS release6.5 x86_64	Nginx Master
6	服务器 6	10.10.1.139	CentOS release6.5 x86_64	Nginx Backup+Mysql+Tomcat
7	服务器 7	10.10.1.144	CentOS release6.5x86_64	Tomcat+Memcache

添加分片用命令 mongo 进行连接 mongos 服务器端口, 然后在 MongoShell 中配置分片信息, 如下所示.

>sh.addShard("shard1/10.10.1.134:27001,10.10.1.135:27001,10.10.1.136:27001")

>sh.addShard("shard2/10.10.1.135:27002,10.10.1.136:27002,10.10.1.137:27002")

>sh.addShard("shard3/10.10.1.134:27003,10.10.1.136:27003,10.10.1.137:27003")

在新建数据库 test 之后, 对数据库 test 开启分片. GridFS 默认使用两种集合 Collection: fs.files 和 fs.chunks 来存储数据, 对集合 fs.chunks 开启分片和索引. 在 Mongo Shell 中的配置如下所示:

>db.runCommand({enablesharding:"test"})

>db.runCommand({shardCollection:"test.fs.chunks",key:{files_id:1,n:1}})

表 3　Mongo 服务器配置

端口	Mongodb 服务器 01(10.10.1.134)	Mongodb 服务器 02(10.10.1.135)	Mongodb 服务器 03(10.10.1.136)	Mongodb 服务器 04(10.10.1.137)
27017	Router Server	Router Server		Router Server
27019	Config Server	Config Server		Config Server
27001	Shard1 副本 1	Shard1 副本 2	Shard1 副本 3	
27002		Shard2 副本 1	Shard2 副本 2	Shard2 副本 3
27003	Shard3 副本 2	Shard3 副本 3		Shard3 副本 1

在服务器 06、07, 也就是 10.10.1.139 和 10.10.1.144 上下载并安装 Tomcat 8. 并将 Java Web 应用打包成 war 包, 然后发布到 Tomcat 服务器的 webapps 目录下. 并在服务器 06 上安装和启动 Mysql. Web 应用使用其作为数据库服务器.

2.2.2 Tomcat 和 Mysql 安装配置

Tomcat 业务服务器集群结构如图 7 所示.

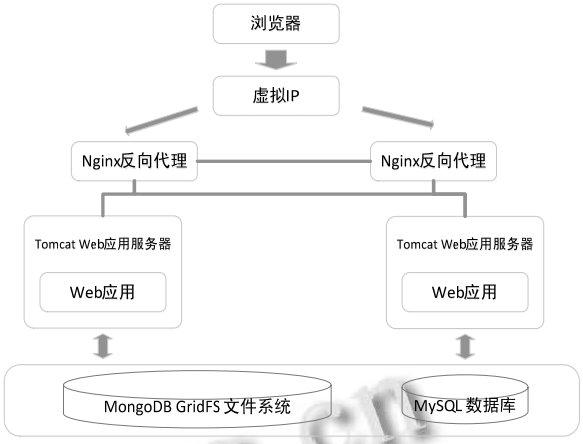

图 7　Tomcat 服务器集群结构

2.2.3 Nginx、Memcached 和 Keepalived 配置安装

在服务器 07 上用 git 下载 nginx-gridfs 插件, 并安装和编译, 命令如下所示:

#./configure --with-openssl=/usr/include/openssl --add-module=../nginx-gridfs/ --prefix=/usr/local/nginx

在 nginx 配置文件中配置负载均衡业务服务器与 MongoDB Mongos IP 地址与端口, 如下所示.

```
upstream dms_pool{
    server 10.10.1.139:8080;
    server 10.10.1.144:8080;
}
location /{
    proxy_pass http://dms_pool;
}
location /test/{
    gridfs test
    field=_id
    type=objectid;
    mongo 10.10.1.137:27017;
}
```

Mencached 是一个高性能的分布式的内存对象缓存系统, 负责业务服务器间共享 session 对象信息, 在服务器 07 上安装和启动. 然后在服务器 06 和 07 的 Tomcat 里配置支持 Mencached 的 session 管理, 即修改文件 server.xml, 添加 Mencached 服务器配置信息.

在服务器 05 和服务器 06 上, 下载安装 Keepalived, 并对 Keepalived 的配置文件 keepalived.conf 进行配置, 在其中设置服务器 05 是主服务器, 而 06 是备服务器. 然后, 在这 2 台服务器上启用 Keepalived 服务. 为了使主服务器 Keepalived 正常运行而 Nginx 出现故障时, 关闭服务, 能够顺利切换, 创建监控脚本,并在 Keepalived 配置文件中实现监控.

3 系统测试

3.1 响应性能测试

对部署在系统中 Web 应用的响应性能用 Jmeter 进行测试, 其中采用的 Web 测试应用的后台技术架构是 SpringMVC3. 对部署在单个服务器上的同一 Web 应用响应性能对比. 分别模拟 30, 50, 80, 100, 120 个用户在 1 秒内发出 HTTP 请求以测试其并发性能, 测量值分别如下表 3 所示. 其中, Samples 表示一共完成了多少个线程, Average 表示平均响应时间,单位是毫秒, Median 表示统计意义上面的响应时间的中值,单位是毫秒, 99%Line 表示所有线程中 99%的线程的响应时间都小于或大于当前数值,单位是毫秒, Min 表示最小响应时间,单位是毫秒, Max 表示最大响应时间,单位是毫秒.

可看到, 随着并发访问数的提高, 优化后的系统的响应时间越少, 吞吐量越大. 然而, 并发访问数提高, 渐渐超过系统的承受能力, 错误率也开始提高. 在并发访问为 80 及 80 以下时, 单服务器响应性能均比优化后的系统表现要好, 这可能是因为还没达到单服务器并发性能极限, 而且, 通过 Nginx 请求转发还需要相应的时间消耗. 在并发访问为 80 左右时, 平均响应时间缩短 9%, 其趋势如下图 8 所示. 这可能是因为, 随着并发访问数的提高, 单服务器响应性能逐渐降低, 而通过 Nginx 的分发和两台服务器分担请求, 单个服务器所需承担的并发数下降, 其服务器响应性能也维持在较高的状态. 而在并发访问数达到 100 时, 单服务器开始出错, 说明已经达到它的并发极限. 而优化后的系统运行良好. 直到并发访问数达到 120, 才达到优化后的系统的并发极限.

表 4 并发访问数据对比

#Samples	30		50		80		100		120	
	单服务器	优化后	单服务器	优化后	单服务器	优化后	单服务器	优化后	单服务器	优化后
Average(ms)	34	37	43	44	57	56	71	64	7748	7528
Median(ms)	33	35	42	43	56	53	60	61	9516	9528
99% Line(ms)	68	76	69	84	88	108	92	98	10028	10089
Min(ms)	30	32	34	38	43	40	43	45	67	32
Max(ms)	68	76	69	84	91	112	1035	229	10046	10099
Error %	0	0	0	0	0	0	0.002	0	0.808333333	0.916666667

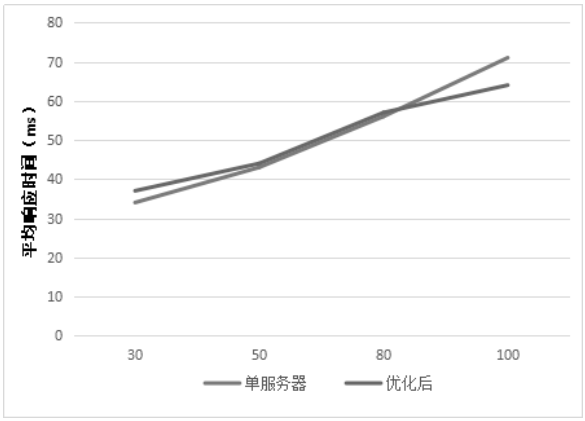

图 7　平均响应时间增长趋势图

3.2 GridFS 文件并发读写性能

3.2.1 GridFS 文件并发写性能

通过 GridFS Java 驱动, 编写 Java 测试应用程序, 在其中创建并发访问线程进行测试, 模拟并发访问十次的平均值, 以减少误差. 然后与 Linux EXT4 文件系统在 500K 和 20M 大小的文件写文件效率进行对比, 并记录其响应时间, 如下图所示. 其中文件都是保存在同一个目录下. 可以看出, 无论是在大文件还是小文件, GridFS 文件上传时间虽然比 EXT4 要慢. 但是, 随着并发数的上升, 文件上传的最大时间是比较稳定的.

上传时间较慢原因有以下几个:第一, 在写文件时, 观察 GridFS 数据库分片状态的变化, 发现文件写入总是在同一个节点上. 这是因为集合 fs.chunks 的分片片键是(files_id, n), 而其中 files_id 是自动生成的. 在这种情况下, 插入总是在一个分片上操作. 第二, 由于 Balancer 的 Chunks 均衡时要锁定资源, 速度较慢, 来不及将新插入的 Chunks 迁移. 第三, 在集合 fs.files 和 fs.chunks 上建立了索引, 对插入速度也有影响. 第四, 其他因素的影响可能导致结果有误差, 如 CPU 资源, 缓存资源的占用等.

3.2.2 GridFS 文件并发读性能

通过 MongoDB GridFS 的 Java API 测试文件的下载效率. GridFS 下载测试是通过 Nginx GridFS 模块在 Nginx 上下载, 而对比的是通过 Tomcat Web 服务器在 EXT4 本地文件系统上下载,在这里搭建了 NFS 文件共享系统方便进行对比. 500K 和 500M 空间大小的文件下载时间如下图 8 所示, 其中文件都是保存在同一个目录下.

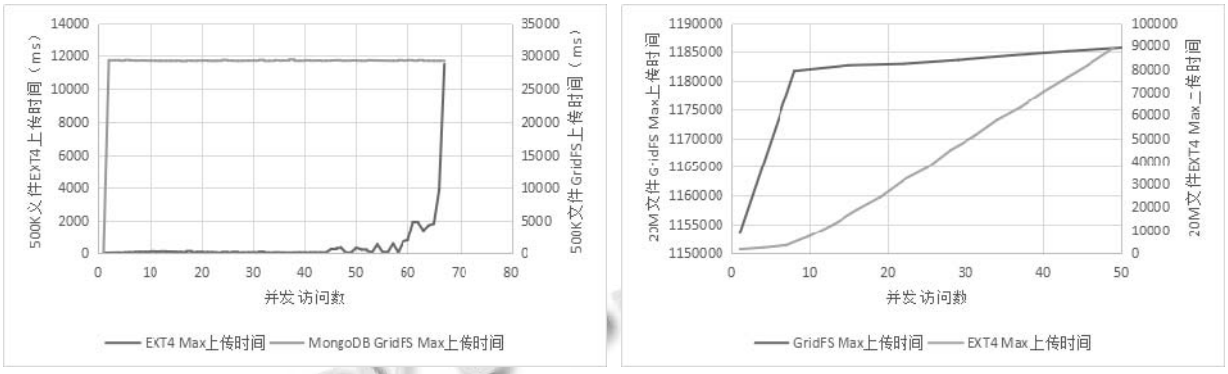

图 8　500K 和 20M 文件并发上传时间

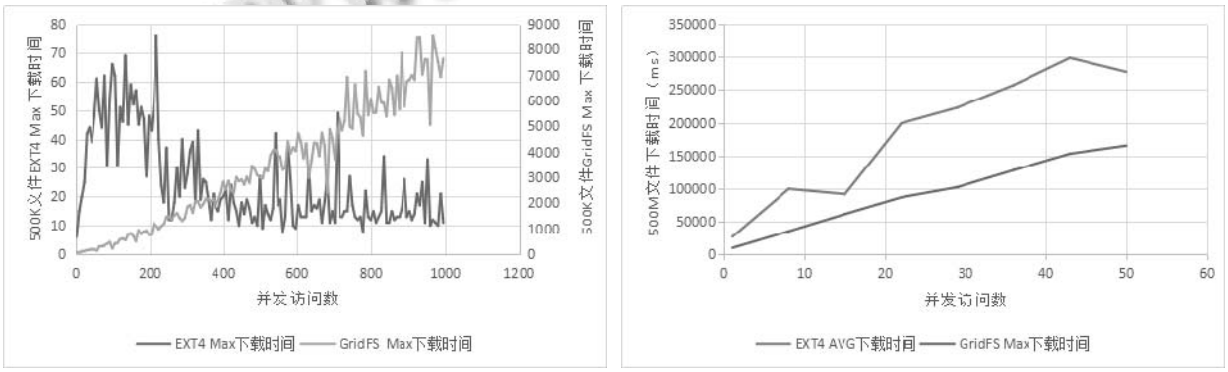

图 9　500K 和 500M 文件并发下载时间

GridFS 的并发查询优势在于通过 Balancer 的策略将数据块 chunks 的查询均匀的分布在各个分片上, 访问负载也随着分散到各个分片上, 从而提高并发性能. 事实上, 当存储的文件数据量少, 由于路由查询等消耗, GridFS 的优势并不明显. 除此之外, 相对于 Linux EXT4 文件系统的单个目录下能存放的文件和文件夹数目有限而且过多的文件数目会导致文件搜索时间过长的问题而言, GridFS 将文件名和路径信息存放在 fs.files 里面则没有这些问题.

文件下载速度与当前网络带宽关系密切, 而网络带宽是随着时间动态变化的. 测试虽然采取多次测量取平均值, 仍然可能导致数据测量有一定偏差.

4 总结

针对不断提高的文件存储需求以及高流量高并发的 Web 应用访问有延迟的问题, 本文提出了一种 Web 优化策略对网站响应性能、文件存储容量、可用性、可靠性和并发读取性能等方面进行提升. 对这种优化的部署架构的响应性能和文件下载速率做了实验和测试发现, 通过 Nginx 系统在高并发情况下有 9%的性能提升. 而且 MongoDB Gridfs 集群的文件上传速率稳定, 下载速率在文件较大的情况下性能较好.

参考文献

1 Chodorow K. MongoDB: The Definitive Guide. O'Reilly Media, Inc. 2013.

2 Fitzpatrick B. Distributed caching with memcached. Linux Journal, 2004, (124): 72–76.

3 Chi X, Liu B, Niu Q, et al. Web load balance and cache optimization design based nginx under high-concurrency environment. 2012 Third International Conference on Digital Manufacturing and Automation (ICDMA). IEEE. 2012. 1029–1032.

4 Liu Y, Wang Y, Jin Y. Research on the improvement of MongoDB auto-sharding in cloud environment. 2012 7th International Conference on Computer Science & Education (ICCSE). IEEE. 2012. 851–854.

5 Jiang W, Zhang L, Liao X, et al. A novel clustered MongoDB-based storage system for unstructured data with high availability. Computing, 2014, 96(6):455–478.

6 Gu Y, Wang X, Shen S, et al. Analysis of data storage mechanism in NoSQL database MongoDB. 2015 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW). IEEE. 2015. 70–71.

7 刘一梦.基于 MongoDB 的云数据管理技术的研究与应用[硕士学位论文].北京:北京交通大学,2012.