Computer Science and Application 计算机科学与应用, 2014, 4, 262-268
Published Online November 2014 in Hans. http://www.hanspub.org/journal/csa
http://dx.doi.org/10.12677/csa.2014.411036

Hans汉斯

Design of Distributed and High Concurrency Log System Based on MongoDb

Shilong Ni[1], Zhentian Lin[1], Qingyuan Cai[2], Haiqiang Xie[2], Rujia Li[2]

[1]FuJian Yirong Information Technology Co., Ltd., Fuzhou
[2]State Grid Electric Power Research Institute, Nanjing
Email: lirujia@sgepri.sgcc.com.cn

Received: Sep. 22nd, 2014; revised: Oct. 20th, 2014; accepted: Nov. 5th, 2014

Abstract

Logging system is an important part of a complete information system. On one hand, massive log information can dig out the common behavior patterns of users to improve service quality; on the other hand, it provides a theoretical basis for system optimization and system structural adjustment. When faced with a high concurrency value system access, timely and accurate collection logs are the key for log analysis. Traditional log records of information have a high coupling with business system code. It affects normal business systems because of taking up system resources. In order to improve this situation, this paper presents a new design way which has zero coupling with an original business system. The way realizes a asynchronous, distributed, parallel processing log records by four parts, which are network request load balancing, distributed cluster services, services in the multi thread processing, and non relational database MongoDb shard expand. We confirmed the feasibility of the design and strong stability by analysised system memory usage and system throughput capacity after tens of millions of data compression test.

Keywords

Distributed, High Concurrency, Large Amounts of Data, Log System

基于MongoDb的分布式高并发日志系统的设计

倪时龙[1]，林振天[1]，蔡清远[2]，谢海强[2]，李汝佳[2]

[1]福建亿榕信息技术有限公司，福州
[2]国家电网电力科学研究院，南京

Email: lirujia@sgepri.sgcc.com.cn

收稿日期：2014年9月22日；修回日期：2014年10月20日；录用日期：2014年11月5日

摘　要

日志系统是一个完整信息系统的重要组成部分，海量的日志信息一方面可以挖掘出用户的通用行为模式，提高系统的服务质量，另一方面还为系统优化，系统结构调整提供了依据，在面对高并发量的系统访问时，及时准确的收集日志成了日志分析的关键，传统日志信息的记录与业务系统代码交织，耦合过高，日志的记录因占用系统的资源进而影响正常业务系统的运行。为了改善这一现状，本文提出了一种与原业务系统零耦合的新设计方式，该方式从网络请求的负载均衡、分布式集群服务、服务中的多线程处理、和非关系型数据库MongoDb的分片拓展四个维度展开，实现了一套异步、分布、并行处理的日志记录系统，后经千万级数据量的压测，从系统吞吐能力，系统内存占用情况等多个方面证实了设计的可行性与强稳定性。

关键词

分布式，高并发，大数据量，日志系统

1. 引言

日志是用来记录用户行为信息和相关服务运行情况的文档，日志的挖掘和分析对于指导系统主要业务的开展和系统运行情况的监控有着重要的意义[1]，日志挖掘的前提是大数据量的日志信息，在面对超高并发量的系统访问时，及时准确的收集日志成了日志分析的关键，大数据时代，日志的记录既要满足高并发的需求，又要与核心业务系统做到零耦合，关系型数据库由于其事务的复杂性，不易拓展性已经不能满足高并发，海量数据的环境。非关系型数据库由于其模式自由便于拓展，在处理高并发写入时有着天然的优势[2]，MongoDb 是非关系型数据库的一种，它使用的面向文档的数据模型，可以自动将数据拆分存储在不同的机器上，这种高效的拓展性，分散了高并发的写入的同时又支持海量数据[3]。在面对巨大的写入并发时，单机版单线程的服务已经很难满足需求，本文旨在实现一种分布式多线程并行写入的日志架构。

2. 设计的核心架构

软件定义下架构实际是利用不同的软件对硬件资源进行再分配，高并发记录日志的核心思想体现在均衡化上，将高并发的请求首先在在网络层进行差分[4]，利用 Ngnix 的负载匀衡功能，将实际的网络请求规则性的差分到不同的服务器组上，应用程序先把请求封装成对象暂时存于内存中，日志系统的底层中多个线程同时获取内存中对象存入 MongoDb 中，然后将内存对象销毁释放内存，在 MongoDb 中也做了相应的分流，利用 MongoDb 的分片技术，将集合分成不同的 chunk 存于多个片中[5]。设计的整体框架图如图 1 所示。

当记录日志的请求来临时，经过纵向的多次拆分与内存的暂存作用，从而解决解决高并发的写入问题，下面将框架中每项技术逐个进行说明。

2.1. 运用 JSONP 实现零耦合设计

传统的日志记录代码内嵌在业务代码中，日志记录与业务处理耦合在一起，存在着效率低，资源争夺，可重用性差等缺点，本次设计将处理日志的系统从主业务系统剥离，构建了一套完全独立的日志系统，日志记录的网络请求有页面端异步发起，但是由于同源策略的限制，XmlHttpRequest 只允许请求当

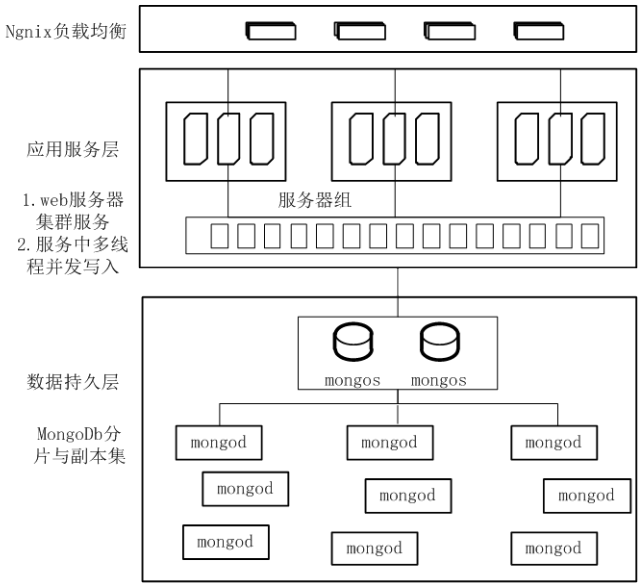


Figure 1. The core framework of log system
图 1. 日志系统核心框架图

前源(域名、协议、端口)的资源，为了解决此跨域的问题，设计中运用了 JSONP 的处理方式，JSONP(JSON with Padding)是一个非官方的协议[6]，它允许在服务器端集成 Script tags 返回至客户端，通过 javascript callback 的形式实现跨域访问。JSONP 核心思想是利用 JS 标签里面的跨域特性进行跨域数据访问，在 JS 标签里面存在的是一个跨域的 URL，实际执行的时候通过这个 URL 获得一段字符串，这段返回的字符串必须是一个合法的 JS 调用，通过 EVAL 这个字符串来完成对获得的数据的处理。

2.2. 运用 Ngnix 实现日志请求的负载均衡

单机版的日志服务无法满足高并发日志的写入，可以将多个日志服务进行集群部署，当请求来临时并不能确定命中哪一台服务器，此时则需要一个专门的代理服务器负责请求的分发。Nginx 是一款高性能的 HTTP 和反向代理服务器[7]，特点是占有内存少，并发能力强，当请求来临时，Nginx 能按照既定规则做路由，把请求分散给到不同的服务上进而达到负载均衡。

常用的 Ngnix 的分发算法有五种，简单轮询，权重轮回，基于 IP 的 hash 算法，基于 url 的 hash 算法和基于 cookie 的分配策略，简单轮询解决了分配公平的问题，但前提是每个服务器的处理能力相同，在实际环境中每个机器的处理能力并不完全相同，权重的轮回考虑到了服务器的能力，因为相同的客户端每次请求的服务器并不同，并不能解决 seesion 的问题，所以在此用的是基于 cookie 的分配策略，首次登陆时给客户端分配一个 cookie，此 cookie 将作为下次访问同一台服务器的验证信息[8]。

2.3. 多线程并行处理

目前高峰期时期对日志服务器网络请求量约 30,000 次/秒，而日志服务器与 MongoDb 资源链接数有限，以往的处理机制是当一个请求来临时，数据库创建一个链接，当处理完成后释放链接以供下次请求的到来，高并发的请求来临时很容易将链接占满从而造成服务的堵塞[9]，为了缓解这一问题，可以将接收的请求以内存资源池的形式进行存储，将计算机内存作为高并发请求和 MongoDb 之间的缓冲[10](图 2)。

多个线程对共享资源进行读写时，会造成资源的争用，为了使多个线程同步执行而又不造成争用，

将共享资源进行规则性的切割，使其变为具体线程的特有资源，这里采用的切割算法是轮询调度。轮询调度算法的原理是每一次把内存资源池中的数据轮流分配给内部中的线程，从 1 开始，直到 M(内部线程个数)，然后重新开始循环[11]。轮询调度算法流程如下：假设有线程 M 个，S = {S1, S2, …, Sm}，一个指示变量 i 表示上一次执行的线程。变量 i 被初始化为 M-1。其代码如下：

```
j = i;
do
{
j = (j + 1) mod m;
i = j;
return Si;
} while (j ! = i);
return NULL;
```

2.4. MongoDb 的分片拓展与副本集

Mongodb 数据库分片是通过并行处理数据实现负载均衡的一种重要手段[12]，同时也是数据库集群实现分布式计算的关键技术。在生产环境中，必须对分片各环节进行合理的架构设计和配置，恰当的分片能提高数据库的计算性能和效率，主备结构的副本集设计能有效的规避单个分片损坏的风险[13] (图 3)。

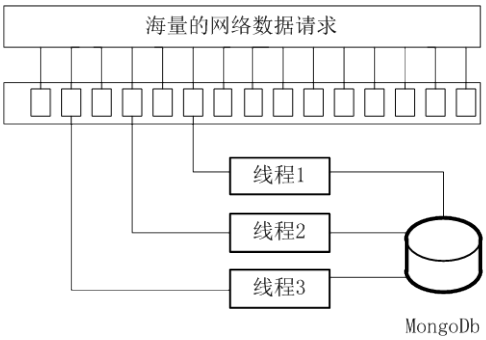


Figure 2. Multi thread parallel processing simulation diagram
图 2. 多线程并行处理模拟图

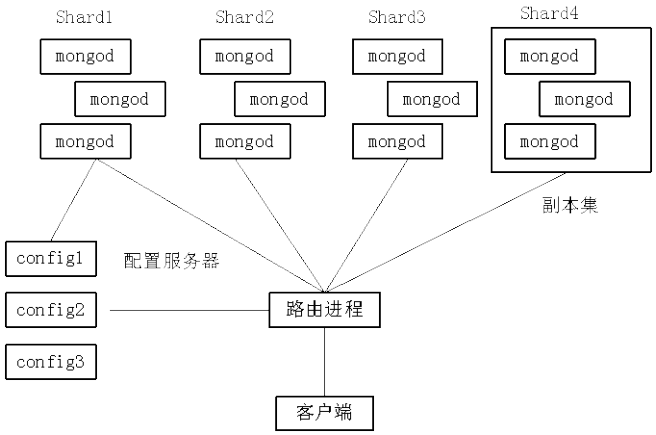


Figure 3. Mongodb slice architecture diagram
图 3. Mongodb 分片架构图

Mongodb 的无规则 schema 使其在高并发，大数据量的环境时很容易进行分布式拓展，shard 是 MongoDb 拓展负载的方式，shard 是将 collection 划分成小的片段 chunks，chunks 有路由进程 mongd 负责存在哪个 shard 上，客户端只需要调用路由进程 mongod，便可以自主进行负载均衡，每个 shard 都有或者多个副本集，当前 shard 出故障时，副本集则立刻替代故障 shard，实现故障的无缝转移[14]。

3. 架构的性能测试

良好的设计的必须以测试为依托，按照上述的设计思想搭建的异步分布式日志采集系统，在面对高并发写入时的表现，最终表现出了良好的性能和稳定的处理能力，因为环境因素的限制，测试采用单个群组节点进行模拟，具体配置如下所述。

3.1. 测试环境的描述

测试环境部署在多个虚拟机中，OS 为 Redhat 5.5 64 位，其中 CPU 是 6 核 12 线程的 Xeon E7-4807，RAM 是 8G 的 ECC DDR3，硬盘是 2 × 300 GB 的 SAS，网络接入千兆交换机，预测速度为 1000 mbps，应用服务器节点采用的 WebLogic，压测工具采用是 Loadrunner，应用服务的具体参数如表 1。

3.2. 测试结果与分析

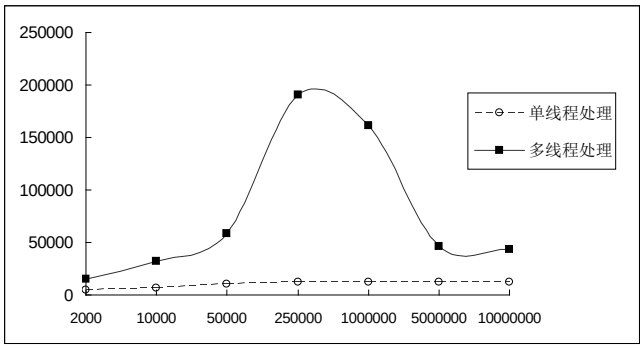
在单个服务中单机 MongoDb 在面对单线程写入与多线程缓冲写入的对比如图 4 所示，从图中可以看出随着网络请求数目的增加，两种方式处理速度均增加，但单线程的处理由于数据库链接资源有限，在增长到一定范围时趋于稳定，多线程处理的速度持续增加，当继续增加到峰值时，单机版 MongoDb 的吞吐能力成了瓶颈，速度开始回收。当网络请求数目在 5 万致 5 百万的范围内时，多线程缓存处理机制明显优于单线程逐条处理。

MongoDb 的分片拓展，由于分散了数据的写入可以适当的提高其吞吐能力，从图 5 可以看出，同样在单机服务多线程处理的条件下，分片后的 MongoDb 在处理网络请求时，开始网络请求的数据少，分片由于其分发机制，在处理速度上并没有优势，当请求数量上剧增时，分片后的 MongoDb 处理速度明显优于单机部署的 MongoDb。

结合现行的主业务系统，系统运行高峰期持续时间 60 分钟，在 60 分钟内预计每秒接收 30,000 个记录日志网络请求，总请求数目约 1000 万条，每条请求封装成内存对象设定为 0.1 K，总数据量 1 G，从图 5

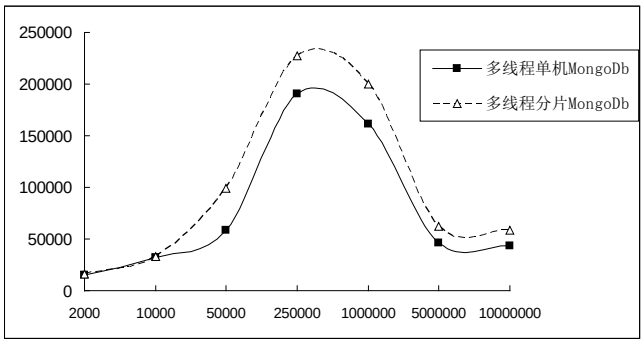
Table 1. Test environment application specific service parameters configuration tables
表 1. 测试环境应用服务具体参数配置说明表

应用服务	具体参数	
Ngnix 反向代理服务	Ngnix 测试版本	Ngnix-1.62
	服务节点数	1 个
Weblogic 应用服务	Weblogic 测试版本	Weblogic-11.1.1.4
	使用 JDK 版本	jdk1.6.0_45
	集群节点数	3 个
	单个服务记录线程数	2 个
MongoDb 数据库服务	MongoDb 测试版本	MongoDb-2.48
	分片个数	3 个
	可提供最大链接数	200 个



X 轴表示当前访问系统的网络请求数，单位是个，Y 轴表示系统处理请求的速度，单位是个/秒

Figure 4. Single MongoDb different threads face network requests comparison chart
图 4. 单服务器单机 MongoDb 不同线程面对网络请求性能对比图



X 轴表示当前访问系统的网络请求数，单位是个，Y 轴表示系统处理请求的速度，单位是个/秒

Figure 5. Multi-threaded MongoDb different shard face network requests comparison chart
图 5. 单服务器多线程不同 MongoDb 面对网络请求性能对比图

中可以看出当网络请求数目在 30,000 时，系统每秒能处理的数目大约在 50,000 个，高峰期能处理的数据量约 1.71 G，从吞吐能力上完全符合数据要求，系统所提供的内存容量为 8 G，即使在高峰期阻塞，8G 的内存依然能提供良好的缓冲作用，从内存角度上不会因内存过高出现宕机现象，符合设计要求。

以上从吞吐能力，内存等多个角度，论述了系统的可行性，优良的系统必须有良好的拓展性和稳定性，在拓展性方面，每个纵向的瓶颈点，都可以通过横向的拓展进行平摊，如网络请求过大可通过添加集群服务，单台处理能力不足可通过适当增加线程，数据库的写入缓慢可通过分片机制，在稳定方面，多个服务节点集群，使其有良好的负载均衡和故障转移能力，MongoDb 分片后做备份与仲裁，保证了数据库的稳定性。综上所述设计的日志收集系统能很好的满足实际业务的需求。

4. 结束语

大数据量日志信息的挖掘分析是未来物联网的一个趋势，通过分析用户层为的规律和系统运行的情况，可以更好的指导主要业务的开展，而信息的收集是分析的基础[15]，本文着重探讨了超高并发日志的准确收集，通过采用请求的负载均衡、服务节点的集群处理、服务中多线程的处理和非关系型数据库 MongoDb 的分片拓展三个方面来实现了一套分布式，高效，与业务系统零耦合的日志收集系统，并对其

性能可靠性进行了评估，证实了架构的可行性。

参考文献 (References)

[1] 何胜韬 (2013) MongoDB 数据库在网络行为分析与控制系统中的应用. *网络安全技术与应用*, **5**, 13-15.

[2] 徐娟娟, 朱成亮 (2011) NOSQL 在 WEB 日志分析中的应用. *信息技术*, **10**, 27-28.

[3] 张文盛, 郑汉华 (2013) 基于 MongoDB 构建高性能网站技术研究. *吉林师范大学学报*(*自然科学版*), **2**, 124.

[4] 庄纪林 (2008) 负载均衡技术在北京大学数字图书馆门户报务中的实践和应用. *现代图书情报技术*, **7**, 74-77.

[5] 胡文焘 (2012) 基于 NoSQL 的空间数据管理系统的设计与实现. 北京中国科学院研究院, 北京.

[6] (2013) Jsonp. http://baike.baidu.com/view/2131174.htm?fr=aladdin

[7] Nedelcu, C. (2010) Nginx HTTP Server. *Packt Publishing*, **5**, 142-144.

[8] 李鹏 (2012) 负载均衡技术在宁夏电力公司信息系统中的实现. *宁夏电力*, **5**, 51-54.

[9] 赵春亮, 张建国, 孟晨 (2011) 基于 Struts2 拦截器的日志记录的设计与实现. *计算机与现代化*, **2**, 150-153.

[10] 王光磊 (2011) MongoDB 数据库的应用研究和方案优化. *信息科技. 中国科技信息*, **20**, 92-94.

[11] (2010) Round-Robin Scheduling. http://blog.163.com/s_u/blog/static/133083672010523310289 4

[12] Banker, K. (2012) MongoDB 实战. 北京人民邮电出版社, 北京, 2-4.

[13] 师德清 (2011) MongoDB 数据库在生产环境中的分片策略研究. *信息与电脑*, **10**, 163-164.

[14] Wang, X.L., Chen, H.P. and Wang, Z.H. (2013) Research on Improvement of Dynamic Load Balancing in MongoDB. 2013 *IEEE* 11*th International Conference on Dependable, Autonomic and Secure Computing*, Chengdu, December 2013.

[15] Gal-Oz, N., Gonen, Y. and Gudes, E. (2011) Security Issues in NoSQL Databases. 2011 *IEEE Workshop on Electronics, Computer and Applications*, Changsha, November 2011.