

文章编号:2096 - 5389(2021)06 - 0106 - 05

四川省气象预警决策发布系统 负载均衡实现与性能优化

淡 嘉,郑 昊,徐 诚,宋雯雯

(四川省气象服务中心,四川 成都 610072)

摘 要:基于四川省气象预警决策发布系统原有架构进行升级,在没有新增服务器和网络设备的情况下,通过 Nginx 服务实现 Web 应用层的负载均衡,减轻网站应对高并发访问的压力。新增两个 Redis 数据缓存服务分别减轻数据库读取压力和提高网关读取海量任务数据的效率。新增消息队列组件实现发布任务至网关模块间的负载均衡和复杂模块的解耦,确保异常中断后的任务恢复。此次升级从软件层面以最小成本实现系统的性能优化,避免因高并发阻塞导致的业务中断。升级后压力测试和实际应用情况表明,系统运行稳定,实现了负载均衡和性能优化的目标。

关键词:负载均衡;Redis;消息队列;预警决策

中图分类号:TP311.52 **文献标识码:**B

Load Balancing Realization and Performance Optimization of Sichuan Meteorological Early Warning and Decision - making Publishing System

DAN Jia,ZHENG Hao,XU Cheng,SONG Wenwen

(Sichuan Meteorological Service Center, Chengdu 610072, China)

Abstract:Based on the upgrading of Sichuan meteorological early warning and decision - making publishing system, the load balance of web application layer is realized by Nginx service without additional server and network equipment, which reduce the pressure of website to deal with high concurrent access. Two new Redis services, one to reduce the database reading pressure, one to improve the efficiency of gateways reading massive tasks. Two new message queues implement load balancing between publishing tasks and gateway, decoupling from complex module, and to ensure task recovery after exceptional interrupt. This upgrade optimizes the performance of the system at the software level at minimum cost and avoids the interruption caused by high concurrent access. The pressure test and actual running proved that the system runs stably, the goals of load balancing realization and performance optimization is achieved.

Key words:load balancing; Redis; message queue; early warning and decision - making

0 引 言

四川省气象预警决策发布系统建于 2013 年,承担全省党政决策用户和运营商定制用户的预警短信发布功能。系统在设计之初,采用传统架构,没有考虑负载均衡能力的建设。随着 2020 年系统推

广至全省 150 余个市县级气象局,短信发布规模进一步扩大,其中党政决策用户增加到 30 万,运营商定制用户稳定在 280 万。发布统计方面,系统在 2018、2019、2020 年分别发布预警短信 6.3 亿、4 亿、5.4 亿条,3 a 累积覆盖决策用户 6.7 亿人次。

近年来,在汛期预警多发时期,大量的预警发

收稿日期:2021 - 03 - 09

第一作者简介:淡 嘉(1986—),男,工程师,主要从事专业气象服务及数据应用工作,E - mail:179233962@qq.com。

布任务同时创建,高并发导致系统网站时常崩溃,数据库阻塞导致性能下降,网关模块大量待发任务堆积,严重影响预警短信的正常及时发布,应急状态下只能切换任务至备机系统释放压力,给值班工作带来巨大挑战。

白龙湖沉船、东方之星客船倾覆等重大事故提醒着全国气象预警工作者责任之大、任务之重,作为预警的传播通道,发布系统的高效稳定对四川省气象预警工作的开展具有重大的意义。

四川省气象预警决策发布系统于 2019 年起进行改造升级,除网络安全等相关升级外,本文介绍软件层面的模块重新设计思路,着重解决系统网站崩溃、数据库读取压力大、网关发送任务堆积这 3 大问题,保障预警发布工作稳定高效开展。

1 系统架构及升级方案

1.1 系统基础架构

四川省气象预警决策发布系统采用 B/S 架构,数据层部署 SQL Server 2012 版本数据库,存储决策用户信息、发布内容、状态报告及系统日志等。应用层部署 Tomcat 容器、运行系统网站,供预警发布工作人员录入预警任务、编辑决策用户信息和查看发送记录统计等。网关接入层对接运营商短信网关,通过模块化组件提供发送预警任务、收集状态报告、实现预警任务缓存等功能,系统架构如图 1 所示。



图 1 四川省气象预警决策发布系统基础架构
Fig. 1 Infrastructure of Sichuan meteorological early warning and decision - making publishing system

1.2 负载问题分析

从发布任务的时间分布角度分析,四川省气象预警集中在汛期,以暴雨为主,预警任务的发布数量呈现“两头低中间高”的分布。系统在每年的 5—9 月处于负载高峰,面临高并发的预警任务创建及发送。如 2018 年 7 月 8 日区域性的暴雨过程,全省

大部分市县在 08 时 20 分左右,集中提交预警发布任务 307 条,覆盖超过 300 万短信用户。

从系统业务逻辑角度分析,如图 2 所示,预警发布人员登录网站页面录入预警任务,审核提交后,网站后端直接访问 SQL Server 数据库提取待发发布号码,并创建发布任务,存放在网关模块的文件缓存中,网关模块实时抓取缓存发送至运营商网关,同时新开线程监听状态报告回执并写回数据库。

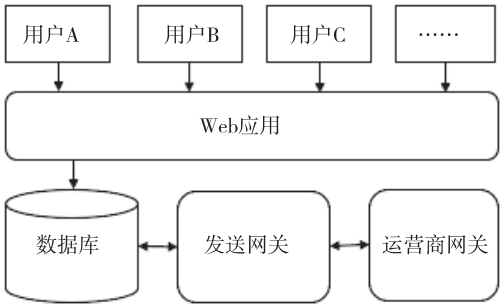


图 2 四川省气象预警决策发布系统业务逻辑(升级前)
Fig. 2 Business logic of Sichuan meteorological early warning and decision - making publishing system (Before upgrade)

这样的业务逻辑存在 3 个问题:①只有 1 个 Tomcat 容器,当网站因为高并发等原因崩溃时,预警发布人员无法及时登录系统发布预警。②结合系统业务逻辑图分析流量高峰时的情况,当大量创建发布任务后,数据库中用户号码表被频繁读取,百万行数据短时间内多次封装输出,而传统数据库面临高并发时性能下降,导致输出效率下降。实际应用中,即使采用索引、分表优化等措施,仍不可避免出现性能问题,问题根源在于传统数据库模式无法应对高并发。③同样是流量高峰的状况下,大量待发任务堆积在网关模块,因为使用文件缓存,磁盘 I/O 效率持续变低,影响发布效率,严重时模块崩溃,读取在内存中的部分任务会丢失,造成漏发。

1.3 系统升级方案

升级后的系统业务逻辑如图 3 所示,系统基础架构保持不变,增加了 3 层架构中的业务模块,调整了主要业务逻辑。

针对之前业务逻辑中存在的 3 个问题,这次升级方案是:①在应用层通过 Nginx 反向代理^[1-4],3 个 Tomcat 容器并行负载。②在数据层增加 Redis 用户缓存作为中间层^[5-7],创建任务请求不直接访问数据库。③在数据层和网关接入层通过消息队列(RabbitMQ)管理并持久化任务列表^[8-9],并在网关接入层增加到 3 个发送网关模块对接运营商短信网关,发送网关中采用 Redis 任务缓存替代磁盘文件缓存。

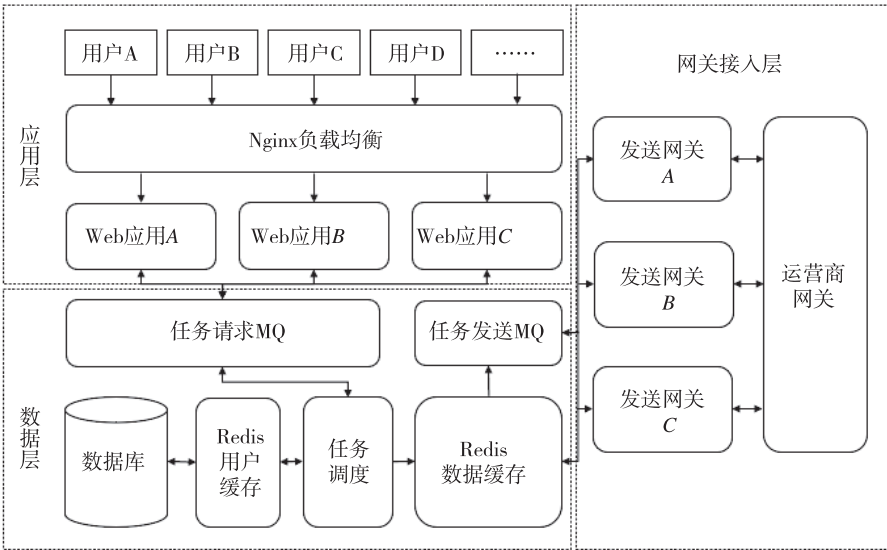


图 3 四川省气象预警决策发布系统业务逻辑(升级后)

Fig. 3 Business logic of Sichuan meteorological early warning and decision - making publishing system (After upgrade)

下面分别从网站应用层、数据层、网关接入层 3 方面详细介绍系统负载均衡策略及性能优化的实现^[10-15]。

2 网站应用层负载均衡

在网站应用层进行负载均衡优化是为了解决因为高并发等原因导致的网站崩溃问题,此次升级中,共部署 3 个 Tomcat 容器,独立运行网站,同时在前端设置 Nginx 服务实现 Web 层的负载均衡。这一过程通过反向代理来实现,Nginx 作为代理服务器,接受来自多个客户端的请求,随后将请求按照既定策略转发给内部网络中不同的服务应用,同时将请求结果返回到客户端。即使其中一个 Web 应用崩溃,访问请求也会被指向到剩余的其它 Web 应用中。这里 Nginx 服务采用的负载均衡策略是轮询发送,目标 IP 的权重为 1,最大连接失败数为 3,连接失败时间为 65 s。

请求会按照顺序发送至 3 个 Web 应用中进行处理,预警录入人员只需要访问网站域名,无需关心 IP、端口情况。当某个应用 IP 在 65 s 内出现连接失败 3 次,请求会被分配至其它 IP,并且在接下来的 65 s 内不会再次请求失败的 IP。Nginx 反向代理服务的逻辑流程图如图 4 所示。

3 数据层性能优化

系统升级前,Web应用直接访问数据库生成发

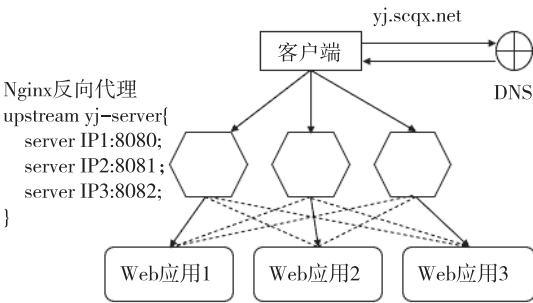


图 4 Nginx 反向代理

Fig. 4 Reverse proxy by Nginx

布任务。根据现代互联网应用的设计指南,直接高并发读取数据库会造成严重的性能问题。因此在升级设计中,增加了 Redis 缓存服务作为数据中间件。Redis 作为 K - V 结构的 NoSQL 数据库,运行在内存中,采用多路复用 I/O 阻塞机制,读取速度较传统数据库更快,针对高并发的支持更加稳定。其次新增消息队列和任务调度模块,保障任务请求的顺序执行和本地持久化。

在 Redis 服务前增加任务请求队列,预警发布人员在 Web 应用中录入发布任务,任务会以消息对象的形式进入请求队列等待任务调度模块处理。在这一步引入消息队列主要基于 3 点考虑:①实现 Web 应用层和数据层的解耦,方便维护和扩展。②利用消息队列中本地持久化特性,即使系统崩溃,

已经生成的发布任务也会缓存在本地磁盘,系统恢复时会自动重新调度数据进入队列,构成预警不漏发的第一道防线。③请求消息队列采用公平分配策略,即同一个任务接收者,在任务处理完毕之前,不会接收到第2个任务分配,保证不会因为分配不均导致的任务堆积。

新增的任务调度模块采用多线程设计,T1 ~ Tn 线程同时处理来自请求队列的多个任务,而限定线程的数量也起到限流的作用,减轻数据层压力。以单个线程为例,T1 线程在接收到第 1 个任务后,访问 Redis 用户缓存提取手机号码,并结合预警内容生成待发布任务,在存入 Redis 任务缓存确认后,生成发送任务对象,提交到任务队列等待网关接入层处理。之后 T1 线程发送确认到请求队列,删除当前任务,并准备接收下一组任务。任务调度模块及请求队列详细的业务逻辑如图 5 所示。

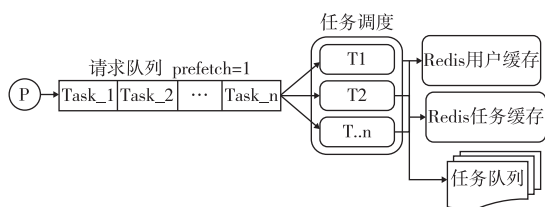


图5 任务调度模块及消息队列业务逻辑

Fig. 5 Business logic of task scheduling module and message queue

4 网关接入层负载均衡

因为运营商短信网关开放的访问速度(即短信下发速度)有限,预警高峰时期,发送任务在网关堆积无法避免,所以针对网关接入层升级的重点在于分散单个网关的发送压力、增加读取任务缓存效率、提升网关模块缓存的稳定性。

首先将负责与运营商短信网关交互的发送网关增加至 3 个,模块间相互独立。当其中一个网关崩溃时,其它两个网关不会受到影响。其次将任务缓存的方式从磁盘文件更换成 Redis 服务,之前方式的弊端在于网关读取磁盘文件时,数据会暂存在内存中,此时任何异常中断,都会导致内存数据丢失引起漏发。升级后利用 Redis 本地持久机制可能保证数据的稳定,提升整个接入网关层的可靠性。同时在 3 个网关前新增 1 个任务发送队列,即使特殊情况下 Redis 数据缓存完全丢失,仍然可以通过任务队列中的信息,重构下发数据,这些机制构成预警不漏发的第二道防线。此外任务队列的根据

既定策略调度任务,调整3个网关的流量,自动分散压力,降低网关因任务堆积而崩溃的可能性。网关接入层业务逻辑如图6所示。

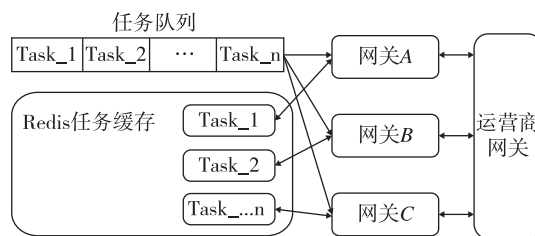


图6 网关接入层业务逻辑

Fig. 6 Business logic of gateway access layer

在数据层经过任务调度模块的处理后,待发送的数据进入 Redis 任务缓存,同时任务队列在得到缓存确认后,通知 3 个发送网关开始对接运营商网关进行下发。以单次任务下发为例,任务队列首先确认发送网关 A 的状态:①如果空闲,网关 A 会接收到第 1 个任务 a ,根据内容在 Redis 任务缓存中提取需要下发的内容,同时对接运营商短信网关进行下发。待下发完成后,确认任务 a 并通知任务队列将其删除。②如果忙碌,任务队列会依次确认下一个发送网关的状态。同理,如果网关 A 在下发任务 a 的过程中异常中断,任务 a 没有得到确认,任务队列会分配任务 a 到下一个网关进行处理。

发送网关与任务队列的业务流程如图 7 所示。

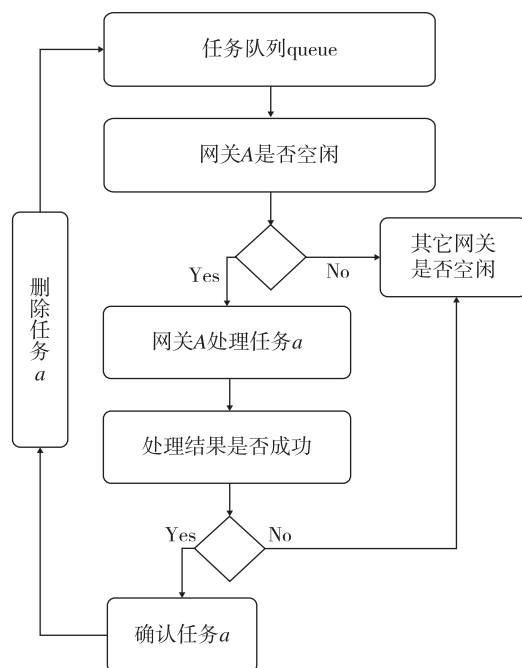


图7 发送网关与任务队列业务流程

Fig. 7 Business process of gateway and message queue

5 系统测试及总结

性能测试以同时提交 500 个任务,平均每个任务包含 6 000 个决策号码,共计 300 万下发为例,与实际应用不同,测试数据不对接运营商网关。连续测试 10 次,基于同一套服务器配置,分别对比系统升级前后,单个任务创建完成平均耗时 a 和网关提取单个任务的平均耗时 b ,其中测试 a 主要指请求决策用户信息构建下发数据,到网关缓存完毕这段耗时,测试 b 考虑下发网关读取任务缓存的耗时。

从测试结果(如图 8)可以看出,测试项目 a 中,升级前平均耗时 2 370 ms,升级后为 276 ms,可以清晰的看出,得益于 Redis 的内存缓存机制,数据读取效率极大提升。测试项目 b 中,升级前平均耗时 277 ms,而升级后仅耗时 79 ms。

负载均衡方面,分析 2020 年 7 月的实际业务统计数据,当月共创建下发任务 57 634 次,发布预警决策短信近 1 亿条,3 个下发网关的负载情况如表 1 所示,之所以网关 A 负载更大,是因为在非高峰下发时段,任务会按照顺序,优先分配给网关 A。当多个任务同时进入队列时,才会均匀负载到其它网关。

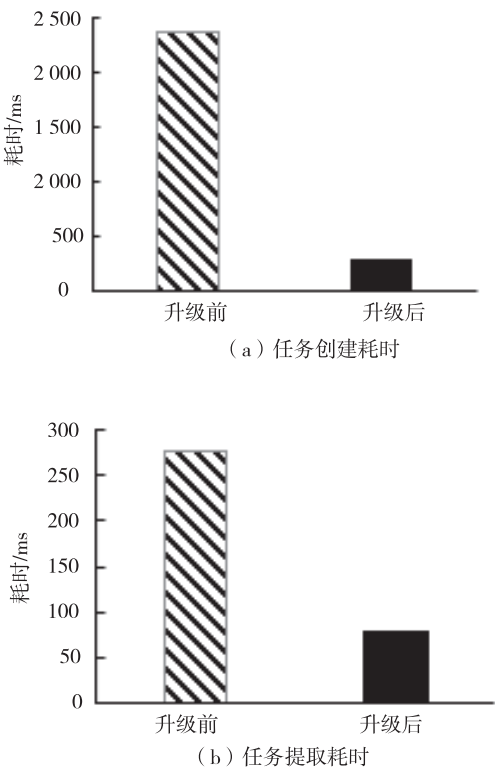


图 8 测试项目耗时对比

Fig. 8 Time comparison of test items

表 1 3 网关在 2020 年 7 月实际下发中的负载情况

Tab. 1 Actual load of three gateways in July 2020

	网关 A	网关 B	网关 C
任务数	23177	19044	15413
短信数(百万)	44.2	37.5	17.8

四川省气象预警决策发布系统在完成负载均衡和性能优化后,提高了网站容错率、数据库及网关的性能得到极大提升,同时系统稳定性也在实际应用得到验证,2020 年没有出现因为系统原因导致的业务中断,保障了预警决策短信下发工作顺利开展。

四川省气象预警决策发布系统此次升级,并没有采用更加完善的数据库集群和硬件级别的负载方案,数据同步机制也不够及时,如果负载进一步提升,系统仍将面临挑战。

参考文献

[1] 林丽丽. 使用高性能 Web 服务器 Nginx 实现开源负载均衡[J]. 大众科技, 2010, 131(7): 37 - 38 + 27.

[2] 王利萍. 基于 Nginx 服务器集群负载均衡技术的研究与改进[D]. 山东大学, 2015.

[3] 怯肇乾. Tomcat 应用服务器高并发优化处理[J]. 电脑编程技巧与维护, 2018(2): 129 - 136.

[4] 王永辉. 基于 Nginx 高性能 Web 服务器性能优化与负载均衡的改进与实现[D]. 电子科技大学, 2015.

[5] 曾超宇, 李金香. Redis 在高速缓存系统中的应用[J]. 微型机与应用, 2013, 32(12): 11 - 13.

[6] 马豫星. Redis 数据库特性分析[J]. 物联网技术, 2015, 5(3): 105 - 106.

[7] 朱进, 胡斌, 邵华, 等. 基于内存数据库 Redis 的轻量级矢量地理数据组织[J]. 地球信息科学学报, 2014, 16(2): 165 - 172.

[8] 任亨, 马跃, 杨海波, 等. 基于 MQTT 协议的消息推送服务器[J]. 计算机系统应用, 2014, 23(3): 77 - 82.

[9] 林许, 张家铭, 杨海波. 基于 MQTT 协议的即时消息业务设计与实现[J]. 计算机系统应用, 2017, 26(3): 219 - 224.

[10] 周莹莲, 刘甫. 服务器负载均衡技术研究[J]. 计算机与数字工程, 2010, 38(4): 11 - 14 + 35.

[11] 李军锋, 何明昕. 高并发 Web 航空票务秒杀系统的设计与实现[J]. 计算机工程与设计, 2013, 34(3): 778 - 782.

[12] 陈关清, 李逸, 杨群, 等. 贵州省气象科技信息共享系统的设计和实现[J]. 中低纬山地气象, 2019, 44(1): 64 - 68.

[13] 王承军. 高并发大数据在线学习系统中的关键技术研究[D]. 中国地质大学, 2015.

[14] 吴和生. 云计算环境中多核多进程负载均衡技术的研究与应用[D]. 南京大学, 2013.

[15] 杨俊萍, 张广通. OpenMediaVault 存储方案在智能网格预报业务中的应用[J]. 中低纬山地气象, 2018, 42(5): 32 - 35.