一种使用 Node.js 构建的分布式数据流日志服务系统①

张 煜

(上海阿尔卡特网络支援系统有限公司 技术项目管理部, 上海 200122)

摘 要: 介绍了使用 Node.js 来实现一个分布式的数据流日志处理系统. 为解决随着运行服务数量的增加, 基于文件的日志操作会带来的存储性能瓶颈和必须要多处查看等问题提供一种方案. 将日志从文件还原为流, 利用 Node.js 来实现一个可分布式获得各日志数据流并且汇聚到中心存储的新型日志系统 octoLogStreamlet. 充分利用 Node.js 的非阻塞 I/O 特性及 NoSQL 数据库的读写高性能, 实现多个数据流同时写入. 系统适用于有集中存放日志需求的计算机服务集群. 集中存放后对日志的处理和使用均提供了极大的便利.

关键词: 数据流; 非阻塞 I/O; 日志; Node.js; Redis

Distributed Data Stream Log Service System Built with Node.js

ZHANG Yu

(Technology and Project management Division, Shanghai Alcatel Network Support System Co.,Ltd., Shanghai 200122, China)

Abstract: This paper introduces a distributed log service System treating logs as data streams built with Node.js. and provides a solution to the problem that with the service number increases, the operation based on files gives rise to disadvantages that files I/O have performance bottleneck and have to be checked at multiple places. Logs are considered as a sort of data stream rather than files. Write a new log service system named octoLogStreamlet which collects the streams to a central storage with Node.js. Combining Node.js' non-blocking I/O model and outstanding performance of NoSQL database, the system meets the requirements to cope with distributed data streams. The system is suitable for computer clusters needing centralized storage of logs, which provides great convenience for processing and utilizing logs.

Key words: data stream; non-blocking I/O; log; Node.js; Redis

传统的服务器后台进程通常使用文件来存放日志, 一般而言会提供缺省的文件路径并允许用户对文件日志存放的位置进行配置[1]. 比如 Nginx 默认的日志就存放在 logs/access.log. 而有些后端服务则允许应用定制自己的日志存放. 典型的代表是 Java 中常部署在 Tomcat 下的 log4j, 应用可以调用其中的 API 接口, 通常还是输出日志到一个文件, 文件还可以滚动写入或者备份成老的日志文件. 所有这些, 使得日志与文件这两个概念往往是结合在一起的.

但是, 随着运行组件的增多, 这种把不同的日志分别存放在服务器本地不同目录的做法会带来一些麻烦和问题. 比如, 如果用Nginx和Tomcat组成一个集群, 假设有 1 个 Nginx, 3 个 Tomcat, 则可能会需要同时查看 4 个日志文件才能完成对系统的访问情况的跟踪. 而且日志的容量和读写速度完全受限于服务器本地硬盘的容量和速度. 这些问题随着系统规模的扩大, 带来的严重性会显著地上升. 比如在一个大规模系统中, 这种日志文件跟随容器的做法是很难实际投入使用的. 这种情况下需要的是可以把日志集中存放, 统一处理和查看.

在上面的背景下, 不妨把日志当作数据流(data stream)来对待, 把日志数据当作一种持续的输出的集合[2]. 文件系统只是用来存放流的一种方式. 而在集群或者多服务程序的环境下, 可以把这些数据流集中汇聚起来. 这样做可以达到两个目的, 集中化的实时查看以及

① 收稿时间:2012-07-23;收到修改稿时间:2012-08-26

© 中国科学院软件研究所 http://www.c-s-a.org.cn

事后查询跟踪. 本文正是提出了这样一个系统, 介绍了设计及实现方案, 并在最后给出了一个应用示例.

1 系统概述

本系统总体分成三部分: 用于采集日志流的客户端 octoLogClient, 处理上行数据的中间层 streamlet, 以及存储节点. 系统组成如图 1 所示.

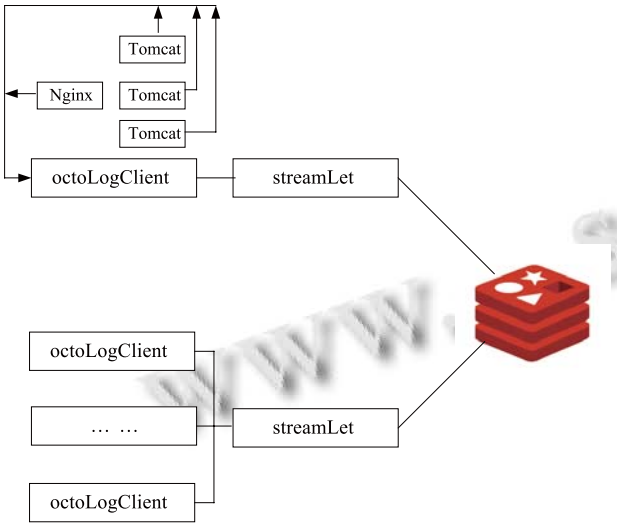

图 1 日志服务系统组成图

其中, octoLogClient 负责读取其他进程的输出日志流, 具体实现是读取一个命名管道, Nginx、Tomcat 等业务进程把日志输出到命名管道. 命名管道是 Linux 系统的一种管道形式, 基于 “先进先出”原理, 与匿名管道的不同在于是使用之前就已经存在, 可以用类似读写文件的方式打开或关闭[3]. 读取到新的日志后, 客户端就把日志以消息的格式发送给上级级联进程 streamlet. 由 streamlet 汇总后写入到数据库中. 需要指出的是, octoLogClient 以及 streamlet 都可以同时运行多个, 共同组建成一个集群. 彼此之间通过设计好的接口实现通信, 从而完成一个分布式的日志服务系统.

2 系统分析及关键技术选择

2.1 octoLogStreamlet 进程组的功能

对日志服务系统的使用对象进行分析, 可以把对象分成两类: 日志输出到文件后台进程, 以及日志输出到标准输出和出错输出流的系统. 前者包括一些遗留软件, 后者则是希望新开发系统标准的日志输出方式.

对采集客户端而言, 如果日志流是直接输出到标准输出流 stdout 的, 那么运行业务进程时可在命令行中把 stdout 重定向到命名管道. 而对于 Nginx 等服务, 则可以在配置文件中直接指定日志文件为命名管道. 通过这种方式, 本日志系统就不会给遗留软件带来任何负担, 应用编写人员也无需掌握新的 API 接口.

当进程作为 server 时, 需要是可以非阻塞和高并发. 所谓非阻塞就是下一级进程访问时, 可以即时把响应回客户端, 而不必等待实际写入动作完成. 尽管这种非阻塞的操作可能存在一定的风险, 比如日志可能最后写入失败了, 但考虑到数据的性质以及对性能的要求, 所获得的效益是可以接受这种风险的. 所谓高并发就是同时能够处理数以万计的连接请求. 因为作为客户端的进程也是并发写入的.

通过对上面这些需求进行分析后, 可以得出实现这个系统的工具需要具备非阻塞、高并发、实时性强的特点. 同时最好还具备跨平台或者可移植性好的特点. 而 Node.js 正符合上述要求.

2.2 Node.js 介绍

Node.js 是一个基于 Google Chrome V8 引擎的 JavaScript 运行环境, 可以在服务器端运行, 从而使得在浏览器中最流行的 JavaScript 语言也可以作为服务来运行[4]. 它充分利用了 JS 语言天生的事件式编程, 以及非阻塞 I/O 的特性. 它所有的库和 API 几乎都首选异步调用的方法.

Node.js 的主要优点:

① 非阻塞的 I/O.

② 回调调用方便, 匿名函数和闭包使得在语言层面具备了异步、事件编程的特性.

③ 受益于 V8, 性能优越运行速度快. 而且只要浏览器的性能竞赛还在继续, 那么 V8 的性能还会继续提升.

Node.js 的主要缺点:

① 作为一个新的运行环境还不成熟, 版本更新很频繁, API 等还在继续变动中.

② 异步编程提高了编程的难度和复杂度.

作为一门表达力极强的语言[5], JavaScript 在浏览器语言的竞争中成为实际上的标准并非仅仅是巧合, 而是它本身确实有巨大的长处. 正是因为 Node.js 具备非阻塞、高并发、实时性强的特点, 而且可移植性好, 再加上适合应用编程人员使用. 经过权衡后, 最终选择了 Node.js 来实现 octoLogStreamlet 日志服务系统.

2.3 存储的需求及数据库选择

日志系统对存储的要求主要在于性能方面需要可

© 中国科学院软件研究所 http://www.c-s-a.org.cn

以大批量处理. 而传统的基于 SQL 的关系型数据库由于要保证事务性, 读写能力方面比新兴的 NoSQL 数据库普遍要差[6]. 考虑到日志的主要用途不是业务性质的数据, 对事务性几乎没有要求, 所以日志系统存储的数据库确定采用 NoSQL 类数据库.

在 NoSQL 数据中, 最终选择了 Redis. 可以把它看成是一个可持久化的内存数据库[7]. 这是一个开源的、高级的 key-value 存储, 支持多种数据结构, 如字符串、hash 组合、列表等[8]. 在处理日志数据中, 列表这种结构是很适合的.

3 系统设计及实现

如系统概述中所描述的, 本系统主要由 octoLogClient、streamlet 以及存储三部分组成.

3.1 总体设计

系统总体设计如图 2 所示:

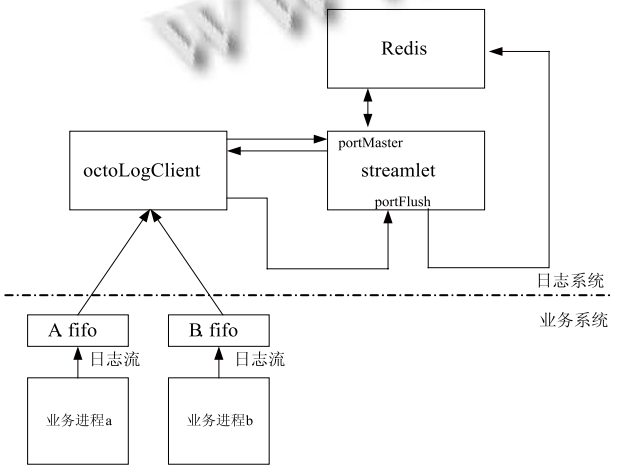

图 2　系统交互设计图

OctoLogClient 启动时, 根据事先配置好的 serviceId 向上级进程 Streamlet 的 portMaster 端口发起获取 tokenId 的请求. 该端口启动一个 HTTP 服务, 收到接入请求后根据输入的 serviceId 产生 tokenId 并保存在 Redis 数据库中, 同时返回 groupId, tokenId, portFlush 等信息, 该内容以 json 格式封装.

随后, 客户端再读取到日志流, 就会通过 portFlush 端口转发给 streamlet. 该端口只接受有效日志信息的转发, 会根据带上来的 tokenId 确定存入到 Redis 对应的 key 中.

3.2 日志流信息读取

如图 2 所示, 这是一个和产生日志的进程运行在同一服务器上的客户端, 并读取事先创建好的 FIFO 命名管道获得业务进程产生的日志数据. 一旦 FIFO 中有数据写入, 则将收到的数据经一定的格式处理后加上 tokenId 信息后发给级联进程.

3.3 streamlet 和 octoLogClient 之间的通信接口

Streamlet 会启动一个 HTTP 服务, 监听在 portMaster 端口, 该协议如下:

操作	GET /token/
参数	serviceId - 服务 id &psId - 进程信息
返回 JSON 信息	{ groupId: - 群 id tokenId: - 令牌 ip: - flush 的地址 portFlush: - flush 的端口 at: - 时间戳 serviceId: - 同请求参数}

此外, 客户端获取 tockenId 后, 还会发日志消息到 portFlush 端口, 该协议直接基于 TCP, 内容如下:

消息	{ token: - 令牌 at: - 时间戳 cdr: - 单条记录 }
返回 JSON 信息	{ result: - 0，正常； -1，出错 }

3.4 数据库存放

启动后收到新的 serviceId 信息, 则以 serviceId:#{ serviceId}作 key, 产生一串 16 位的随机字符串 groupId 作 value 进行存放; 同时, 再产生一串 16 位的随机字符串作 tokenId, 并以 token:#{tokenId}为 key, groupId 为 value 进行存放; 最后将 groupId:{#groupId}为 key, serviceId 作 value 也保存起来. 即 serviceId/groupId, tokenId/groupId 和 groupId/serviceId 三个一对一的关联.

以上三者的数据类型均是字符串. 在收到增加日志的请求后, 先根据 token 找出 groupId, 然后以 log:group:#{groupId}为 key, 而 value 的类型则采用列表格式, 将新增的日志添加到列表的末尾.

4 应用示例

部署的视图可参考图 1, 在 Linux 系统上模拟监视一个 Nginx 和三个 Tomcat 日志的效果. Nginx 在收到请求后, 会把请求轮流转发给后端的 3 个 Tomcat.

4.1 系统部署

第一步, 在/tmp 目录下创建 4 个命名管道, 分别为 nginx.fifo, tomcat1.fifo, tomcat2.fifo 和 tomcat3.fifo.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

第二步，在 Nginx 的配置文件 nginx.conf 中，加入 access_log /tmp/nginx.fifo，使得 Nginx 的日志会输出到这个管道中. 此次模拟中采用 Nginx 1.2.1 版本. 同时让 Nginx 监听在 80 端口.

第三步，配置 Jdk 和 Tomcat，采用 Jdk1.6 及 Tomcat7.0.29. 在 Tomcat 的 catalina.sh 脚本中设置变量 CATALINA_OUT=/tmp/tomcat1.fifo. 对于另外两个 Tomcat 也如法炮制，分别设为 tomcat2.fifo 和 tomcat3.fifo. 同时将 3 个 Tomcat 的监听端口分别设置为 8001, 8002 和 8003.

第四步，配置 octoLogCliet，增加要监听的命名管道及服务名称：{serviceId: "lottery", psId: "Nginx", pipe: "/tmp/nginx.fifo"}, {serviceId: "lottery", psId: "Tomcat1", pipe: "/tmp/tomcat1.fifo"}, {serviceId: "lottery", psId: "Tomcat2", pipe: "/tmp/tomcat2.fifo"}, {serviceId: "lottery", psId: "Tomcat3", pipe: "/tmp/tomcat3.fifo"}

第五步，安装并启动 Redis 数据库，此处采用 2.4.14 版本，监听在默认的 6379 端口.

第六步，配置 portMaster 为 10000, portFlush 为 10001，并启动 streamlet 组件以及 octoLogClient.

第七步，配置 Nginx 对 Tomcat 转发请求:

```
upstream tomcat {
   server 127.0.0.1:8001;
   server 127.0.0.1:8002;
   server 127.0.0.1:8003;
}
location ~* \.(jsp|do|action)$ {
   proxy_pass http://tomcat;
}
```

第八步，启动 Nginx 及 3 个 Tomcat 进程.

4.2 测试及结果

当通过 80 端口访问/basictest.jsp 这些 Tomcat 服务的内容时，Nginx 会转发给后端的 3 个 Tomcat，则 Tomcat 连同 Nginx 均会产生日志. 日志首先会输出到相应的管道中，随后由日志流系统汇聚到数据库中. 集中后的结果可通过 Web 界面的方式查询. 如图 3 所示就是查询的结果. 根据记录下的 psId 可以列出是哪一个进程写入的，而时间戳则可以对日志进行排序. 即便是运行多个服务进程的情况下，通过不同的查询键 log:group:#{groupId}，也能迅速分类找出内容. 一旦日志集中存放后，查找及处理的便利性都有了极大的提高. 对于整个系统的运行情况也有了更全面的了解，极大地提高日志查询效率.

图 3 日志查询结果

5 结语

本文所提出并设计实现的日志系统，将分布在各系统中的日志流汇聚到了 Redis 数据库中. 日志数据是每个实际投入使用系统的重要组成部分. 传统的基于文件的日志处理形式在面临集群及服务增多的情况下存在明显的不足. 整个日志系统的架构考虑了可扩展性，几个模块均可以规模化水平扩展，以满足不同规模集群的处理要求. Node.js 的使用确保了该日志系统可以在并发能力和响应时间上满足实际应用的需要.

通过组合经典的 FIFO 命名管道技术和新生的事件式非阻塞编程的 Node.js 以及 NoSQL 数据库 Redis，就可以实现这样一个日志服务系统.

参考文献

1 内梅特.Unix/Linux 系统管理技术手册.第 4 版.北京:人民邮电出版社,2012.

2 宫学庆,闫莺,周傲英,等.数据流处理技术在电信网管系统中的应用.计算机科学与探索,2008,2(2):180–191.

3 Stevens WR.UNIX 网络编程 2:进程间通信.第 2 版.北京:清华大学出版社,2002.

4 Tilkov S, Vinoski S. Node.js: Using JavaScript to build high-performance network programs. Internet Computing, IEEE, 2010,14(6):80–83.

5 Dahl R.Node.js.http://nodejs.org/.

6 StoneBraker M. SQL databases v.NoSQL databases. Magazine Communications of the ACM, 2010,53(4):10–11.

7 杨艳,李炜,王存.内存数据库在高速缓存方面的应用.现代电信科技,2011,12:59–64.

8 Sanfilippo S, Noordhuis P. Redis. http://redis.io/.

© 中国科学院软件研究所 http://www.c-s-a.org.cn