基于 Tokyo Cabinet 的分布式存储系统

Pudge Project Design Document

Version : 2.0.0

2013-01-15

DOCUMENT HISTORY

Ed.	Version	Author	Change
1	1.0.0 Draft	王磊，陈传生	增加了基本内容和架构图
2	2.0.0	王磊，陈传生	增加序列图

目　录

1 项目概述

1.1 项目目标

基于 Tokyo Cabinet 实现 key-value 数据的存储功能，能用命令行实现对远程数据库的 open、close、get、put 和 delete 等操作。

客户端能够连接远程数据库服务器进行操作；服务器端能够支持大并发量请求和较好的容错性；系统能够充分利用分布式优点，并有一定的扩展性和容错性。

1.2 功能特点

- 采用一致性哈希算法和冗余存储机制使得服务器的变动对客户端影响减小。
- 服务器端使用 I/O 复用和多线程的方式提高并发性能。
- 服务器端使用 libevent 库进行 I/O 复用，以隔离平台相关性。
- 优良的架构设计使得负载均衡在各服务器上，不依赖单台服务器性能。
- 能够自动获取 IP 和可用端口，简化操作。

2 基本工作原理和软件架构

2.1 基本工作原理

2.1.1 分布式策略

本系统使用一致性哈希算法进行数据分布式存储，一致性哈希算法相对于普通哈希算法能够较好的应对服务器变更所造成的影响，保证系统的可用性。

一致性哈希将 Key 和服务器节点编号一起映射到一个 0 ~ 2^{32}-1 的环形空间上，如下图：

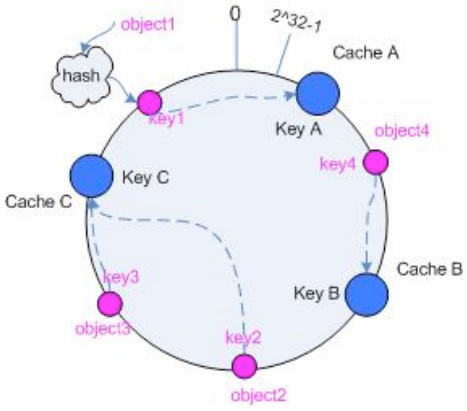

图 1 一致性哈希

考虑到系统的特性，我们这里将服务器的 IP 地址和端口号作为关键字进行 HashPJW 运算，并使用线性探测的碰撞解决方法给每个服务器分配一个唯一的 ID，该 ID 对应于环空间

的某一点，当需要对某对象进行操作时，使用对象的 Key 在环空间中进行顺时针探测，第一个遇到的节点即为该对象应该归属的服务器节点。

我们的一致性哈希算法使用循环链表实现，从链表首节点开始，直到遇到第一个 ID 大于对象 Key 的服务器节点，该节点即为归属节点，若到链尾仍没有找到，则链表第一个节点为归属节点。

根据一致性哈希的特点，为了保证当某个节点失效后的数据可用性，客户端在存储某个对象时，除了将对象存储在对应的归属节点外，也将对象冗余存储在归属节点的后续几个节点上，当归属节点失效后，存储的数据不会丢失，仍然可用。

2.1.2 服务器状态维护策略

考虑到新加入某个服务器或者失效某个服务器时，客户端需要获得及时的更新以保证正常运转，所以需要在系统中较好地维护和传递各服务器状态。

我们使用的策略为：

（1） 系统中有一个专门进行服务器状态维护的 Master 服务器，Master 服务器启动后，其他数据库服务器启动并与 Master 服务器连接，Master 更新在线服务器列表并将该列表 PUSH 到各个数据服务器上；Master 与各数据服务器间维持一个“心跳”连接，若某服务器失效，Master 通过“心跳”连接得知服务器失效，将该服务器从在线服务器列表中删除并将更新后的列表 PUSH 到各个数据服务器上。

（2） 客户端启动时，先与 Master 服务器连接，获得最新服务器列表，将该列表生成一个一致性哈希链，并在该链上进行查找节点等操作。客户端需要进行数据库操作前，首先随机选取一个服务器，从该服务器得到最新服务器列表，若该列表与客户端先前存储的列表相同，则不进行任何操作，否则对列表进行更新，对一致性哈希链进行更新。

2.1.3 服务器软件的并发处理策略

服务器端需要支持大并发量，所以我们的 Pudge 在数据服务器和 Master 服务器都使用了 I/O 复用和多线程结合的技术。

由于各个系统环境下对于 I/O 复用的实现各不一致，如 Linux 下为 EPOLL，UNIX 下为 KQUEUE，所以为了隔离平台相关性，我们使用了开源库 libevent 进行 I/O 复用，libevent 实质上是对 EPOLL 这种结构进行了封装，使得能够根据不同的平台使用不同的 I/O 复用机制。Libevent 将套接字描述符作为事件放入 event 池中，当有消息到达某套接字时，libevent 触发相应的回调函数，进行处理。

在数据服务器端，使用多个请求处理线程，每个线程都有各自的消息队列，当 libevent 调用回调函数后，回调函数将该请求随机分配给某个线程，将消息放入线程消息队列，并激活相应线程信号量，线程便可以开始处理。

在 Master 端，对于每一个与服务器间的心跳连接使用一个线程，该线程负责对该服务器进行监视、删除和更新 PUSH 服务器列表工作。

2.2 软件架构设计

2.2.1 系统总体架构图

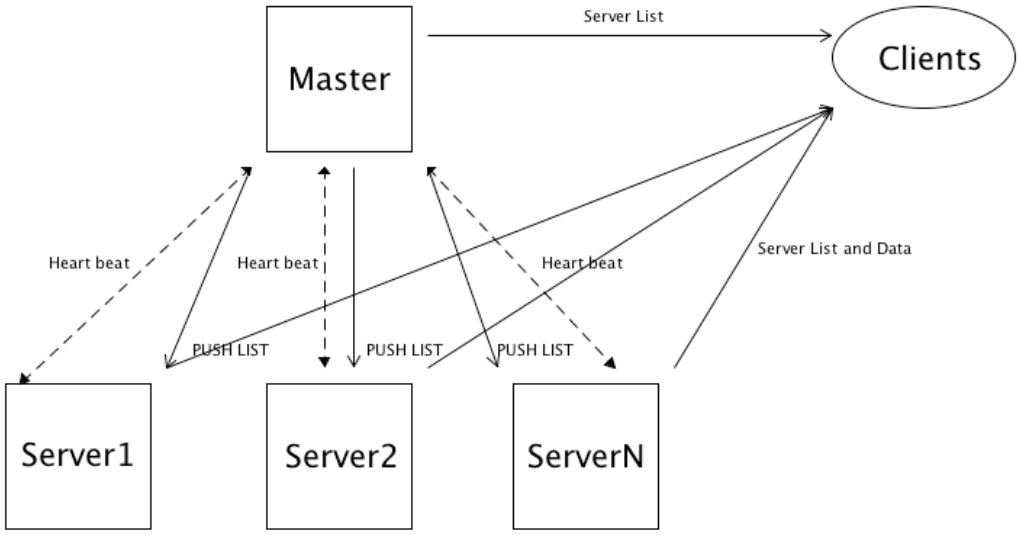

图 2 Pudge 系统架构图

2.2.2 系统模块分解图

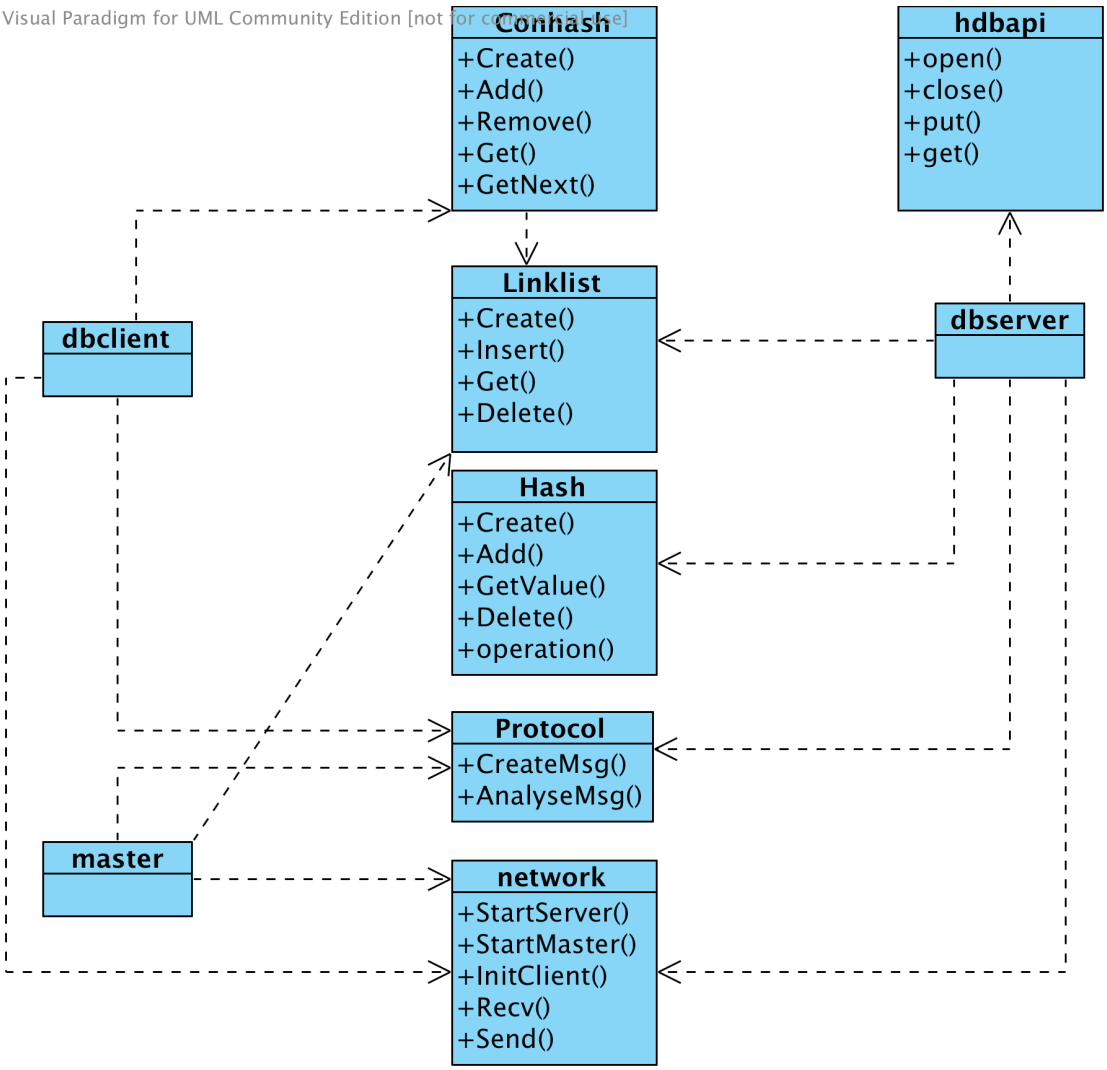

图 3 系统模块分解图

2.2.3 系统交互序列图

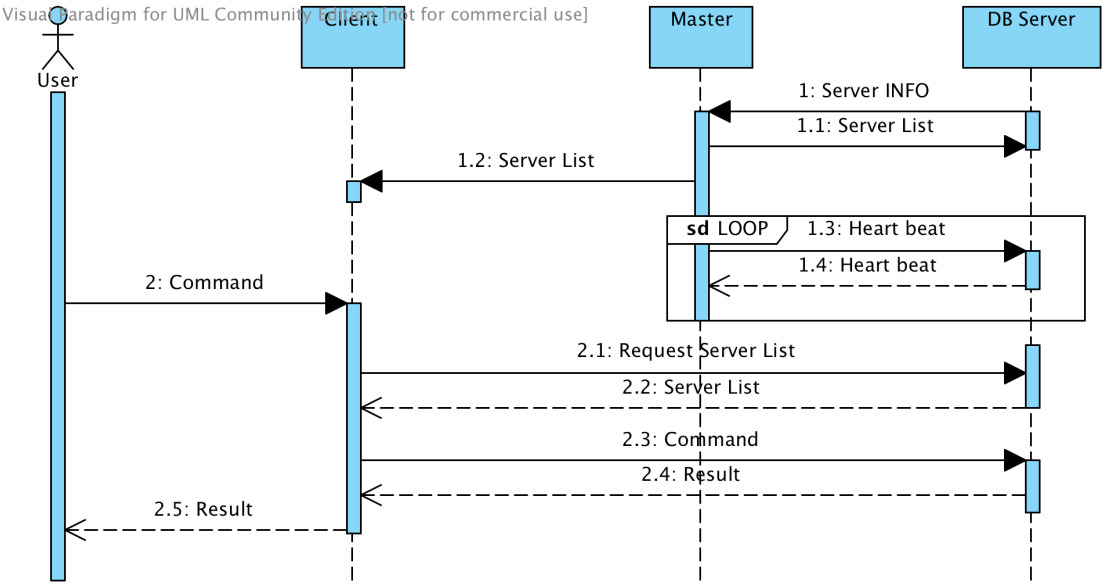

图 4 系统交互序列图

3 相关接口定义说明

3.1 Protocol 协议的封包结构

Cmd_code(4)	Size1(4)	Data1(Size1 bits)	Size2(4)	Data2(Size2 bits)

Pudge protocol packet format

- Cmd_code(4): 消息类型
  - OPEN 1
  - CLOSE 2
  - PUT 3
  - GET 4
  - DELETE 5
  - EXIT 6
  - OPEN_OK -1
  - CLOSE_OK -2
  - PUT_OK -3
  - GET_OK -4
  - DELETE_OK -5
  -
  - NEW_SERVER 100

- GET_SERVER_LIST 101
- SERVER_LIST_OK 102
- MASTER_OK 199
- HEART_BEAT 800
- HEART_BEAT_OK 801
- UPDATE_SERVER_LIST 1000
-
- ERROR 0

- Size1(4bits): 数据 1 的大小
- Data1(Size1 bits): 数据 1 的内容
- Size2(4bits): 数据 2 的大小
- Data2(Size2 bits): 数据 2 的内容

4 参考资料

【1】“分布式哈希”和“一致性哈希” http://stblog.baidu-tech.com/?p=42

【2】 Programmer’s Toolbox Part 3: Consistent Hashing
http://www.tomkleinpeter.com/2008/03/17/programmers-toolbox-part-3-consistent-hashing/

【3】 Programming with Libevent http://www.wangafu.net/~nickm/libevent-book/

【4】 Linux Sokcet 编程 http://www.cnblogs.com/skynet/archive/2010/12/12/1903949.html