分布式小文件系统SDFS (small distributed file system)

需求及设计文档

作者：黄晓萌

## 项目背景

当前logserver只有一台机器存储日志，会造成以下问题：

1. 随着日志越来越多，一台机器将来肯定无法存储所有日志。
2. 多个worker往一台logserver写日志，会造成生产者大于消费者，消费不过来的情况。
3. 多个客户端从一台logserver读日志，也会造成logserver压力过大。
4. logserver是单点，不可靠

## 可能的解决方案

那么是否可以使用现有的分布式文件系统来替换logserver呢？

首先从现有jarvis的需求来说：

1. worker从shell的标准出错获取日志，一行一行写到logserver中，即要满足append操作；
2. ironman和jarvis执行任务的时候，可以流式打印日志，即要满足可以根据offset和size随机读；
3. 有些日志（某个任务废弃了，或者几个月前的日志），可以定期清理，即要满足删除操作；
4. 执行日志和结果总体来说不是很大，即单机肯定能存得下，但是数量会特别多。

我们选取了一些分布式文件系统进行对比，如下表：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 操作  文件系统 | 顺序写 (append) | 随机读 (offset+size) | 删除 | 小文件存储 |
| HDFS | ✔ | ✔ | ✔ | ✖ |
| HBASE | ✖ | ✖ | ✔ | ✔ |
| MOSS | ✖ | ✖ | ✔ | ✔ |
| KAFKA | ✔ | ✖ | ✖ | ✔ |

表1

从上表可以看出，HDFS最满足我们的需求，但是HDFS不适用小文件。我们现在一天1W多个任务，算上失败重试的估算到2W个，每个任务有两个日志（标准输出和出错日志），即每天有4W个小文件，一个月下来120W个，一年下来就1000多万了。居安思危，如果以后业务扩张的比较快，一天有10W个任务呢？这会对HDFS的namenode造成很大的压力，显然不是非常好的解决方案。

## SDFS概要设计

小文件：在单机能存得小的文件叫做小文件。

在HDFS中，存储的最小单位是block；而在SDFS中，存储的最小单位是文件。

在SDFS中，主要有3个组件，分别是namenode, datanode, client，总体设计图如下：

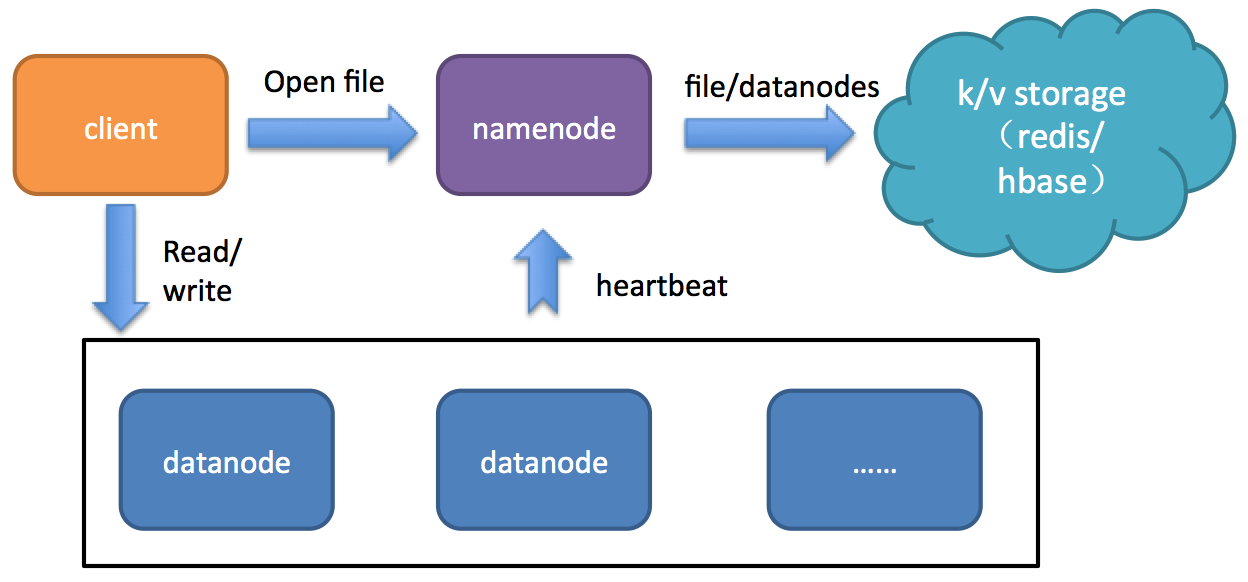


图1

如图1所示：

1. Namenode作为SDFS的metastore，管理文件和datanode的映射关系。不同于HDFS把文件的元数据通过inode树的方式存储在自己的内存中维护文件和Datanode的映射关系，SDFS把文件和datanode的映射关系存储在可水平扩展的基于K/V的存储中（比如redis或者hbase）。
2. Datanode作为SDFS的存储节点，会定期向namenode发送心跳，汇报自己的存储使用率。
3. Client作为SDFS的客户端，封装了对namenode和datanode的访问，提供类似于本地文件系统一样的接口。

下面以redis作为kv storage描述一下SDFS读写文件的流程

1. SDFS写文件



图2

如图2所示，一开始调用client的open接口，namenode从redis上没有找到文件，则说明是新建文件，从维护的datanode列表中选取一个返回给client，并存储到redis中；

client拿到datanode的地址，就可以不用和namenode通信了，直接和datanode通信，append数据；

当client数据写完之后，往datanode发送close消息（datanode会往该文件中写一个结束符），然后释放句柄。

1. SDFS读文件



图3

如图3所示，client调用open接口，namenode从redis上找到该文件所在的datanode，返回给client;

之后和写文件差不多，拿到datanode的地址，就直接和datanode通信。

## SDFS功能介绍

1. 除了基本的文件读、写、seek等操作，还支持上传、下载等操作
2. 支持海量小文件存储，namenode和datanode都支持水平扩展
3. 支持按照多级分区存储（类似于hive的partition），比如一级分区是username，二级分区是调度日期，则存储的时候会存储类似为…/{username}/{date}/taskid\_attempid.err。

这样的好处是，比如某个用户离职了，他的任务都需要删除了，那么日志根据用户名来删除将会很简单。

1. 支持一级分区的定期清理（类似于kafka的retention机制，允许某一个topic的日志保留多少时间）
2. 基于user/group的ACL权限控制
3. rebalance，新增加一个节点，支持rebalance