分布式文件系统概述

3170103240 张佳瑶

一、 概述

分布式文件系统是指文件系统管理的物理存储资源不一定直接连接在本地节点上，而是通过计算机网络与节点相连。本地系统只能访问与主机通过 I/O 总线直接相连的磁盘上的数据。局域网出现之后，各台主机间通过网络互连起来，提出了文件共享的需求，即一台主机需要访问其他主机的磁盘。分布式文件系统因此诞生。而如今，数据量迅速增长，数据在规模上不断增大，数据的种类也不断丰富。为处理大规模的异构化的数据，大多数大数据解决方案都建立使用分布式文件系统之上。

分布式文件系统基于客户机/服务器模式而设计。数据保存在服务端，客户端像访问本地文件系统一样访问位于远程服务器上的文件。为提高读写性能和系统可扩展性，客户端通常对数据进行缓存。缓存和一致性产生矛盾。一个网络内可能包含多个可供用户访问存储资源的服务器。分布式文件系统存在对等特性，一些系统在扮演客户端的角色的同时也在扮演服务端。

分布式文件系统的要求有：大容量、高性能、可扩展性、高可用性、可管理性。文件的内容被复制到不同的磁盘、磁盘控制器、或主机，因此可以提高文件系统的可用性。通过配置大数据文件块，可以实现高速的顺序读写。

二、 发展历程

分布式文件系统的发展经过网络文件系统、共享存储集群文件系统和基于对象存储的并行文件系统的历程。第一代的分布式文件系统一般以提供标准接口的远程文件访问为目的，更多关注访问的性能和数据的可靠性，以 NFS（Network File System）和 AFS（Andrew File System）最具代表性。第二代的分布式文件系统面向广域网，追求大容量，以 XFS、Tiger Shark 为代表。第三代文件系统的研究与应用得益于网络技术的发展和普及。数据容量、性能和共享的需求使得这一时期的分布式文件管理系统的规模更加庞大复杂。有更多先进技术得以实现、应用：分布式锁、缓存管理技术、Soft Updates 技术、文件级的负载平衡等。第四代文件系统的研究目标是结合 SAN 和 NAS 两种体系结构，充分利用两者的优势。

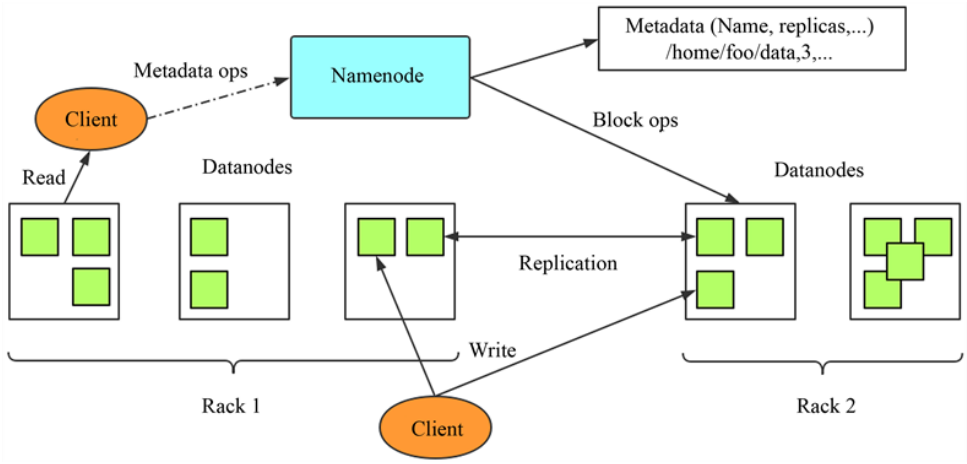
三、 元数据

元数据时描述文件属性的数据，包括目录内容，文件大小以及文件块指针。元数据保留了从文件名到记录位置的映射，可用于在数据存储节点之间检索文件数据。在 EB 级别的文件系统中，元数据的规模可以达到 PB 级，因此不可忽视元数据对系统性能的影响。基于哈希的映射和子树分区是主流的元数据分区方案。基于哈希的映射通常将文件的路径名或其他标识符映射到一些哈希键中，并通过从元数据键到 MDS 键的投影将元数据分配到 MDS。基于哈希的策略可以在 MDS 之间均匀分配文件元数据，因此该方案可以平衡系统工作负载。与基于哈希的映射不同，子树分区保留了名称空间树的结构。与基于散列的映射相比，它提供了更好的局部性和更大的 MDS 独立性，而工作负载可能无法在 MDS 之间均匀分配，这很容易产生“热点”。此外，随着各个子树随着时间的增长或收缩，它需要系统管理员的干预才能在 MDS 上重新分配或手动重新平衡元数据存储负载。

四、 HDFS

典型的分布式文件系统，以 Hadoop 为例。Hadoop 是一个分布式系统基础架构。Hadoop 的一个核心子项目是 Hadoop 分布式文件系统，是一个基于 Java 的文件系统。HDFS 具有高容错性的特点，设计用来部署在通用硬件上。充分利用集群告诉运算和储存的优势，以高传输速率来访问应用程序的数据，可以流的形式访问文件系统中的数据，非常适合大规模数据集上的应用。对于一致性问题，采用一个“一次写入多次读取”文件访问模型。这个模型简化了数据一致性问题，一个文件经过创建、写入和关闭之后就不需要改变。

HDFS 文件系统采用主从系统结构。一个提供元数据服务的 Namenode 节点和若干个提供数据存储的 Datanode 节点构成一个 HDFS 集群。分布式文件系统将元数据和应用数据分别存储在 Namenode 及 Datanode 中，各个服务器之间通过 RPC 协议进行通信。HDFS 命令空间是分层次的文件及目录结构。Datanode 及 Namenode 之间通过心跳信息进行通信，在通信中报告自己的运行状况。



图表 1 HDFS **基本架构**

分布式文件系统 HDFS 被广泛部署为许多并行大数据分析的基础。Apache Hadoop 是 HDFS 的开源实现。Apache Hadoop 在 HDFS 上运行，具有一个 NameNode。Hadoop 集群中有一个或多个 DataNode。NameNode 是管理整个集群的节点，DataNode 是存储数据并处理作业的节点。NameNode 中的 JobTracker 管理作业调度，TaskTracker 在 DataNodes 中处理分布式作业任务。客户接收随机选择的 DataNode 列表用于存储块文件副本的候选 DataNode。然后，客户端将其数据块刷新到第一个 DataNode 节点。第一个 DataNode 开始以小尺寸接收数据部分（例如 4 KB），将每个部分写入其本地存储库并将其传输到列表中的第二个 DataNode。第二个 DataNode 开始接收和写入该部分到其存储库，然后将其刷新到第三个 DataNode。与使用条带化策略均匀分布数据的传统并行文件系统不同，在存储节点之间，数据密集型文件系统（例如 HDFS）存储每个数据单元并具有多个副本。基于相对随机策略，可能导致存储节点之间的数据分布不均匀；基于数据检索，在 HDFS 策略中，存储节点包含的数据越多，可以选择存储节点提供数据的可能性就越高。检索时来自 HDFS 的数据，客户端进程将首先尝试读取来自运行磁盘的数据。如果需要数据不在本地磁盘上，则该过程将从包含所需数据的另一个节点。

Hadoop 可以在 Ubuntu 上部署。配置好 Hadoop 集群之后，在 DataNode 上启动 Hadoop 集群。

```
root@hadoop1:~# start-dfs.sh
WARNING: HADOOP_SECURE_DN_USER has been replaced by HDFS_DATANODE_SECURE_USER.
SECURE_DN_USER.
Starting namenodes on [hadoop1]
hadoop1: namenode is running as process 7394.  Stop it first.
Starting datanodes
hadoop3: datanode is running as process 1233.  Stop it first.
hadoop4: datanode is running as process 1171.  Stop it first.
hadoop2: datanode is running as process 1204.  Stop it first.
Starting secondary namenodes [hadoop1]
hadoop1: secondarynamenode is running as process 7621.  Stop it first.
root@hadoop1:~# start-yarn.sh
Starting resourcemanager
Starting nodemanagersStarting nodemanagers
hadoop2: nodemanager is running as process 1313.  Stop it first.
hadoop4: nodemanager is running as process 1280.  Stop it first.
hadoop3: nodemanager is running as process 1342.  Stop it first.
root@hadoop1:~# mr-jobhistory-daemon.sh start historyserver
WARNING: Use of this script to start the MR JobHistory daemon is deprecated.
WARNING: Attempting to execute replacement "mapred --daemon start" instead.
cc^H^Hroot@hadoop1:~# jps
7394 NameNode
8340 ResourceManager
7621 SecondaryNameNode
8758 Jps
8719 JobHistoryServer
```

图表 2 启动集群

在各个节点上，查看启动的进程，DataNode 和 NodeManager 进程已启动。

```
root@hadoop2:~# jps
1313 NodeManager
1810 Jps
1204 DataNode
```

图表 3 查看节点进程

当 Hadoop 的 DataNode 正常启动，live datanodes 不为 0，集群成功启动。

```
Live datanodes (3):

Name: 10.173.32.15:9866 (hadoop2)
Hostname: hadoop2
Decommission Status : Normal
Configured Capacity: 21103222784 (19.65 GB)
DFS Used: 24576 (24 KB)
Non DFS Used: 3152748544 (2.94 GB)
DFS Remaining: 17053372416 (15.88 GB)
DFS Used%: 0.00%
DFS Remaining%: 80.81%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Xceivers: 1
Last contact: Fri Sep 27 17:15:06 CST 2019
Last Block Report: Fri Sep 27 17:13:30 CST 2019
Num of Blocks: 0


Name: 10.173.32.16:9866 (hadoop3)
Hostname: hadoop3
Decommission Status : Normal
Configured Capacity: 21103222784 (19.65 GB)
DFS Used: 24576 (24 KB)
Non DFS Used: 3152736256 (2.94 GB)
DFS Remaining: 17053384704 (15.88 GB)
DFS Used%: 0.00%
DFS Remaining%: 80.81%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Xceivers: 1
Last contact: Fri Sep 27 17:15:06 CST 2019
Last Block Report: Fri Sep 27 17:14:06 CST 2019
```

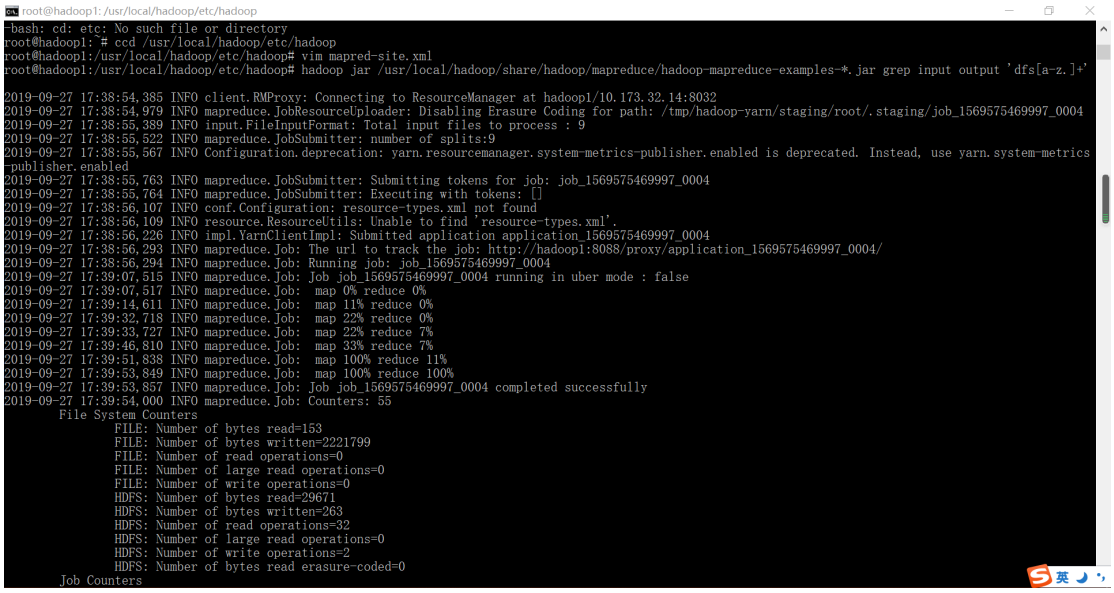
图表 4 集群正常启动结果

首先创建 HDFS 上的用户目录：将/usr/local/hadoop/etc/hadoop 中的配置文件作为输入文件复制到分布式文件系统中：

```
hdfs dfs -mkdir -p /user/root/input
```

```
hdfs dfs -put /usr/local/hadoop/etc/hadoop/*.xml /user/root/input
```

运行测试程序后的结果如下图所示：



图表 5 **操作** HDFS

五、 其他

Ceph 是基于 Linux-btrfs 文件系统的分布式文件系统，以 C++实现，性能卓越。它具有强大的容错能力，可以解决单点故障问题，支持在线扩容和冗余备份。Ceph 有强大的扩展能力，基于 CRUSH 算法，没有中心节点，对象的元数据分布在各台物理机上。

Lustre 是一个大规模的、安全可靠的，具有高可用性的平行集群文件系统，可以支持超过 10000 个节点，适合存储小文件、图片。Lustre 文件系统中，客户端生成一个全局存储空间，用户数据通过客户端存入 lustre 文件系统中，所以客户端的设置也会影响系统的性能。Luster 底层存储设备采用通用存储设备，大部分采用 RAID 方式，既能保证聚合存储容量，又能提供数据保护。

fastDFS 是以 C 语言开发的一个开源的轻量级分布式文件系统，采用了 client-tracker-storage 结构，能实现 RAID 10 相似功能，但不能在不同计算机之前迁移。FastDFS 由跟踪服务器、存储服务器和客户端构成。追踪服务器负责接收客户端的请求，存储服务器负责实际存储数据，客户端负责上传数据。

TFS 是淘宝采用的分布式文件系统，主要针对存储海量的非结构化数据。在 TFS 文件系统中，NameServer 负责管理文件元数据，通过 HA 机制实现主备热切换。所有的元数据存储在内存中，因此处理效率非常高效。Dataserver 作为分部署数据存储节点，具有负载均衡的作用。

六、 总结

分布式文件存储不仅提高了存储空间的利用率，还实现了弹性扩展，降低了运营成本，避免资源浪费，适合如今数据高速累积的时代场景。

七、 引用

[1] 杜振南, 朱崇军. 分布式文件系统综述[J]. 软件工程与应用, 2017, 6(2): 21-27.

[2] J. Lee, B. Kim and J. Chung, "Time Estimation and Resource Minimization Scheme for Apache Spark and Hadoop Big Data Systems With Failures," in IEEE Access, vol. 7, pp. 9658-9666, 2019.

[3] D. Huang et al., "Achieving Load Balance for Parallel Data Access on Distributed File Systems," in IEEE Transactions on Computers, vol. 67, no. 3, pp. 388-402, 1 March 2018.

[4] Y. Gao, X. Gao, X. Yang, J. Liu and G. Chen, "An Efficient Ring-Based Metadata Management Policy for Large-Scale Distributed File Systems," in IEEE Transactions on Parallel and Distributed Systems, vol. 30, no. 9, pp. 1962-1974, 1 Sept. 2019.