大数据技术及应用

第三章 分布式文件系统GFS


Google
——GFS

提纲

Google文件系统GFS

- 系统架构
- 容错机制
- 系统管理技术

Google业务

- 全球最大搜索引擎、Google Maps、Google Earth、Gmail、YouTube等

秘密武器：云计算平台！

数据量巨大，且面向全球用户提供实时服务

Google云计算平台技术架构

- 文件存储，Google Distributed File System，GFS
- 并行数据处理MapReduce
- 分布式锁Chubby
- 分布式结构化数据表BigTable
- 分布式存储系统Megastore
- 分布式监控系统Dapper

Google设计GFS的动机

- Google需要一个支持海量存储的文件系统
  - 购置昂贵的分布式文件系统与硬件？

是否可以在一堆廉价且不可靠的硬件上构建可靠的分布式文件系统？

GFS 设计背景

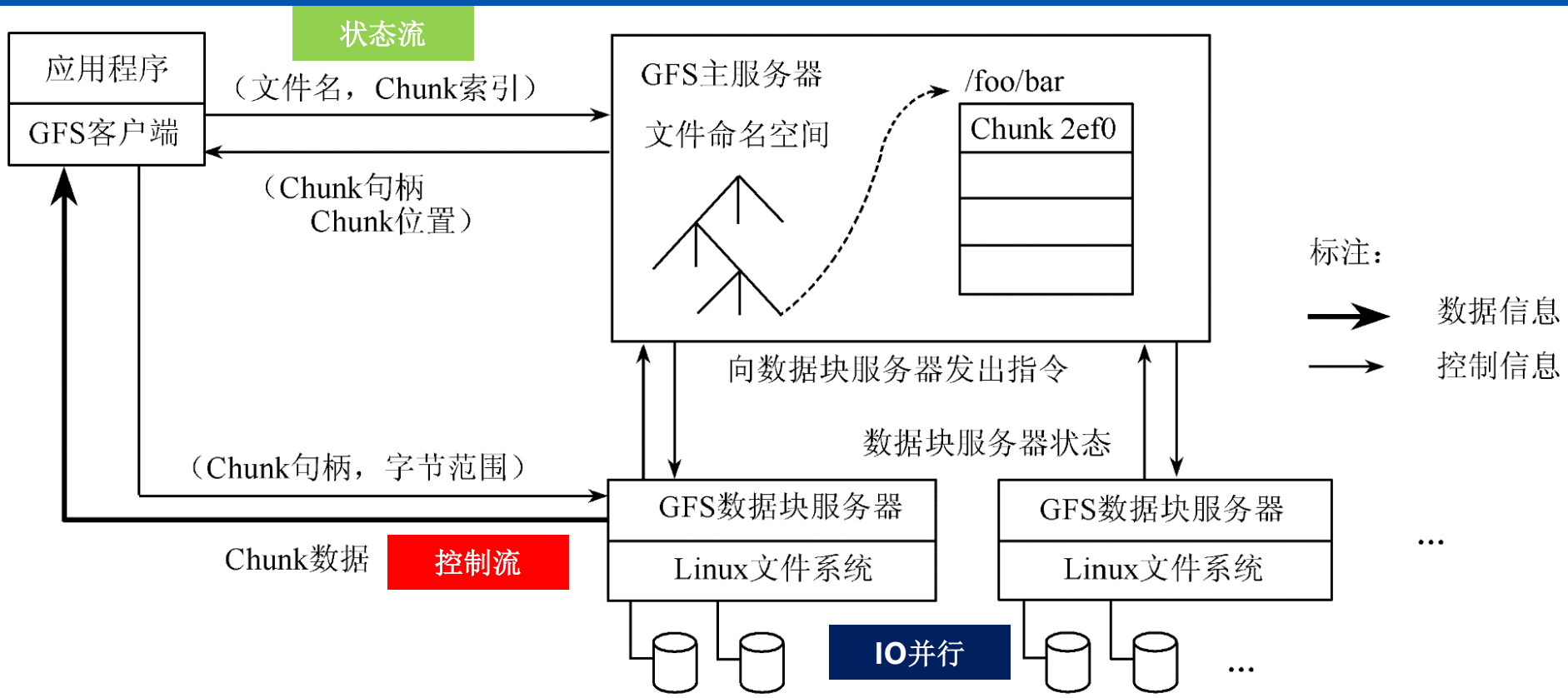
- 设计预期
  - 持续监视、错误检测、容错处理、自动恢复
  - 对大文件有效管理同时支持小型文件
  - 文件超大文件的顺序写入、随机小规模的写入
  - 大量的操作为在文件后追加数据，几乎没有随机写入，写完后只读，且读取方式基本上只有大规模顺序读和小规模随机读
  - 支持多路合并模式进行操作
  - 高性能的稳定带宽的网络要比低延时更加重要
- 接口
  - 常见操作 create、delete、open、close、read、write
  - 特殊操作 snapshot、record append(原子操作)

文件系统

GFS架构是怎样的？

不间断的数据存储服务

GFS系统架构



Client（客户端）：应用程序的访问接口

Master（主服务器）：管理节点，在逻辑上只有一个，保存系统的元数据，负责整个文件系统的管理

Chunk Server（数据块服务器）：负责具体的存储工作。数据以文件的形式存储在Chunk Server上

GFS特点有哪些？

实现机制

- 客户端首先访问Master节点，获取交互的Chunk Server信息，然后访问这些Chunk Server，完成数据存取工作。这种设计方法实现了控制流和数据流的分离。
- Client与Master之间只有控制流，而无数据流，极大地降低了Master的负载。
- Client与Chunk Server之间直接传输数据流，同时由于文件被分成多个Chunk进行分布式存储，Client可以同时访问多个Chunk Server，从而使得整个系统的I/O高度并行，系统整体性能得到提高。

GFS特点

采用中心服务器模式

- ▶可以方便地增加Chunk Server
- ▶Master掌握系统内所有Chunk Server的情况，方便进行负载均衡
- ▶不存在元数据的一致性问题

不缓存数据

- ●文件操作大部分是流式读写，不存在大量重复读写，使用Cache对性能提高不大
- ●Chunk Server上数据存取使用本地文件系统，若读取频繁，系统具有Cache
- ●从可行性看，Cache与实际数据的一致性维护也极其复杂

在用户态下实现

- □利用POSIX编程接口存取数据降低了实现难度，提高通用性
- □POSIX接口提供功能更丰富
- □用户态下有多种调试工具
- □Master和Chunk Server都以进程方式运行，单个进程不影响整个操作系统
- □GFS和操作系统运行在不同的空间，两者耦合性降低

只提供专用接口

- ○降低实现的难度
- ○对应用提供一些特殊支持
- ○降低复杂度

GFS特点

GFS Architecture

Client

Client代码包含了google文件系统的API，并且会和master和chunkserver进行通信。

- client和master通信—交互元数据信息
  client会缓存从master获取的元数据信息，以便对同一块的操作不在通过client-master交互。

- client和chunkserver通信—交互文件数据
  client所有的数据相关的通信是直接和chunkserver进行，但是不会缓存文件数据。

GFS特点

GFS Architecture

Master

- Master负责管理所有的文件系统的元数据
  - 文件和chunk的namespace
  - 访问控制信息
  - 文件到chunk的映射关系
  - 当前chunk的位置等等

- Master控制系统级别的活动
  - chunk的分配管理
  - 孤点chunk的垃圾回收机制
  - chunk在chunkserver之间的移动

- Master用心跳信息周期地跟每个chunkserver通信，给他们以指示并收集他们的状态
- 单一Master设计必须减小master的读/写操作，以避免它成为集群瓶颈。

13

GFS特点

GFS Architecture

Metadata

- Master保存三种类型的数据：
  - 文件和chunk的namespace
  - 文件到chunks的映射关系
  - 每个chunk副本的位置

- 所有的元数据都是保存在Master的内存里。
- 前两个由在master本地硬盘的记录所有变化信息的operation log来持久化保存的，这个记录也会在远端机器上保存副本。
- Master并不持久化保存chunk位置信息，启动地时候以及chunkserver加入集群的时候，向每一个chunkserver询问他的chunk信息。

GFS特点

GFS Architecture

Chunkserver

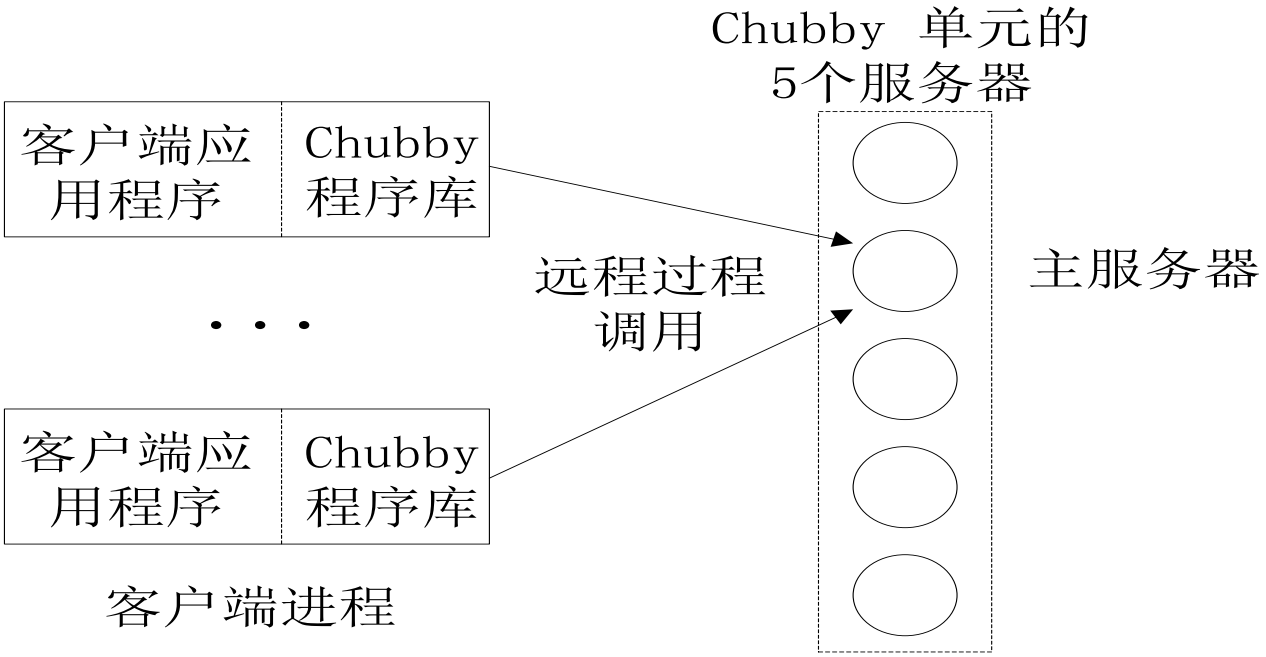
- 在GFS下，每一个文件都拆成固定大小的chunk(块)。每一个块都由master根据块创建的时间产生一个全局唯一的以后不会改变的64位的chunk handle标志。
- chunkservers在本地磁盘上用Linux文件系统保存这些块，并且根据chunk handle和字节区间，通过Linux文件系统读写这些块的数据。
- 出于可靠性的考虑，每一个块都会在不同的chunkserver上保存备份。缺省情况下，我们保存3个备份。

分布式锁服务Chubby

- 主要用于解决分布式一致性问题
  - 在一个分布式系统中，有一组的Process，它们需要确定一个Value。于是每个Process都提出了一个Value，一致性就是指只有其中的一个Value能够被选中作为最后确定的值，并且当这个值被选出来以后，所有的Process都需要被通知到
- 粗粒度的分布式锁服务
  - Chubby是Google为解决分布式一致性问题而设计的提供粗粒度锁服务的文件系统
  - 其他分布式系统可以使用它对共享资源的访问进行同步

分布式锁服务Chubby的系统架构

在架构上，Chubby集群一般有5台机器组成，每台机器都有一个Replica(副本)，其中有一个Replica会被选为Master节点，Replica在结构和能力上相互对等，Replica使用Paxos协议来保持日志的一致性，Replica都有可能离线，然后重新上线。重新上线后，需要保持与其它节点数据的一致。Client端使用Chubby的客户端库来访问。

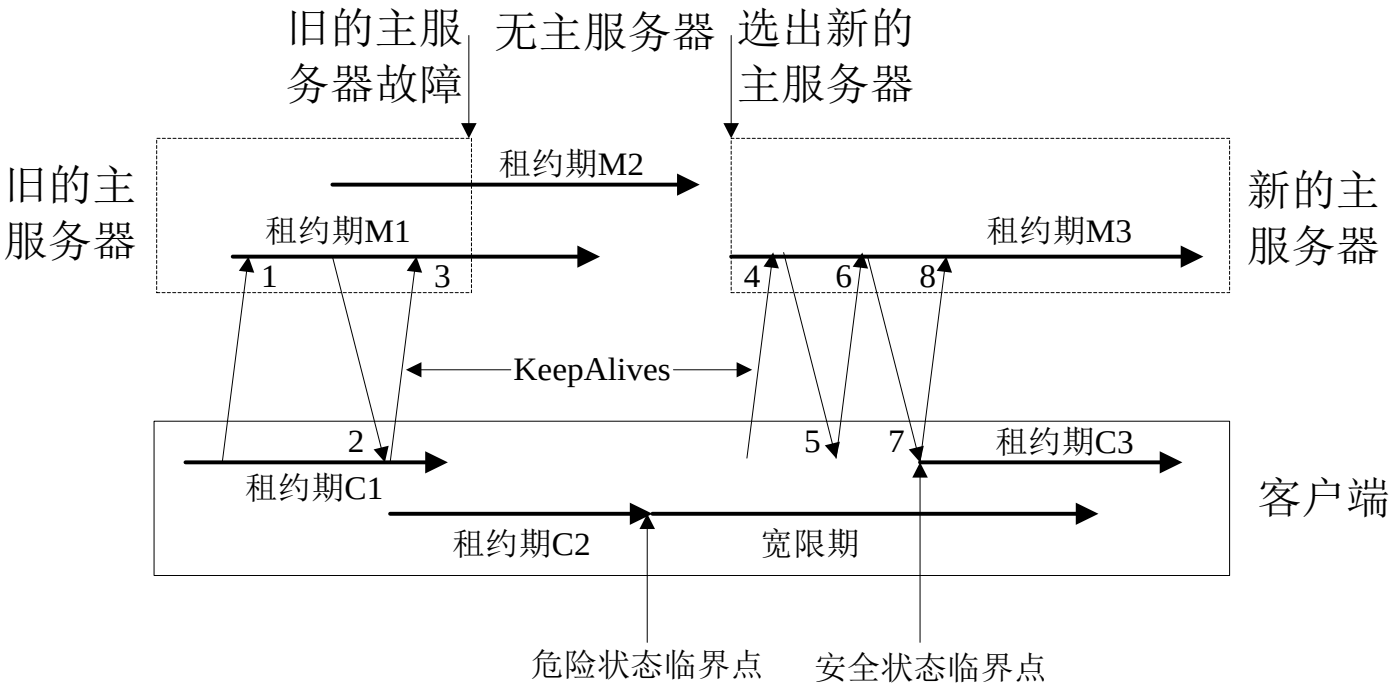


分布式锁服务Chubby的文件系统

- Chubby系统本质上就是一个分布式的、存储大量小文件的文件系统
  - Chubby中的锁就是文件
  - 在GFS的例子中，创建文件就是进行“加锁”操作，创建文件成功的那个server其实就是抢占到了“锁”
  - 用户通过打开、关闭和存取文件，获取共享锁或者独占锁；并且通过通信机制，向用户发送更新信息

Client 与Chubby的通信

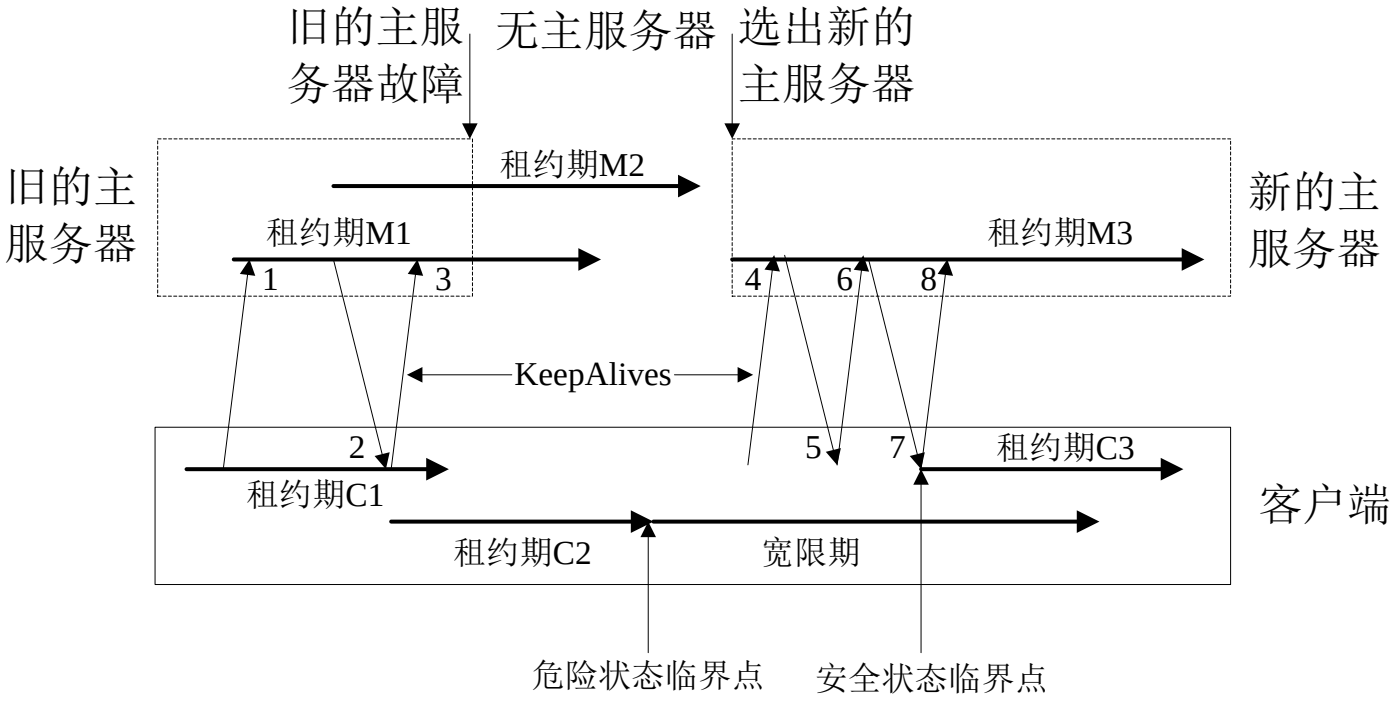
Chubby 会议用来保持主服务器与客户端之间的联系。每段会议持续一段时间，并通过keep alive的握手机制维持。除非chubby客户端通知主服务器，否则在会议有效期间，客户端的句柄，锁服务和缓存数据一直维持有效。保持有效和出错回调是客户端和主服务器通信的重点，



- 在通信部分，需要考虑的主要问题是session与lease，以及客户端与服务器端需要传递保存的信息。
- Session和lease是维持客户端与服务器端通信的标识，严格来说session是一个虚概念，真正起作用的实体是lease。Lease通过定期的发送keep-alive RPC来保持Client和Master的连接。
- 在Master端，由于需要为每个连接的Client设置一个session并且保存client打开的文件handle，所以将client打开的handle保存在session里，这样可以根据client很容易定位到其打开的handle。

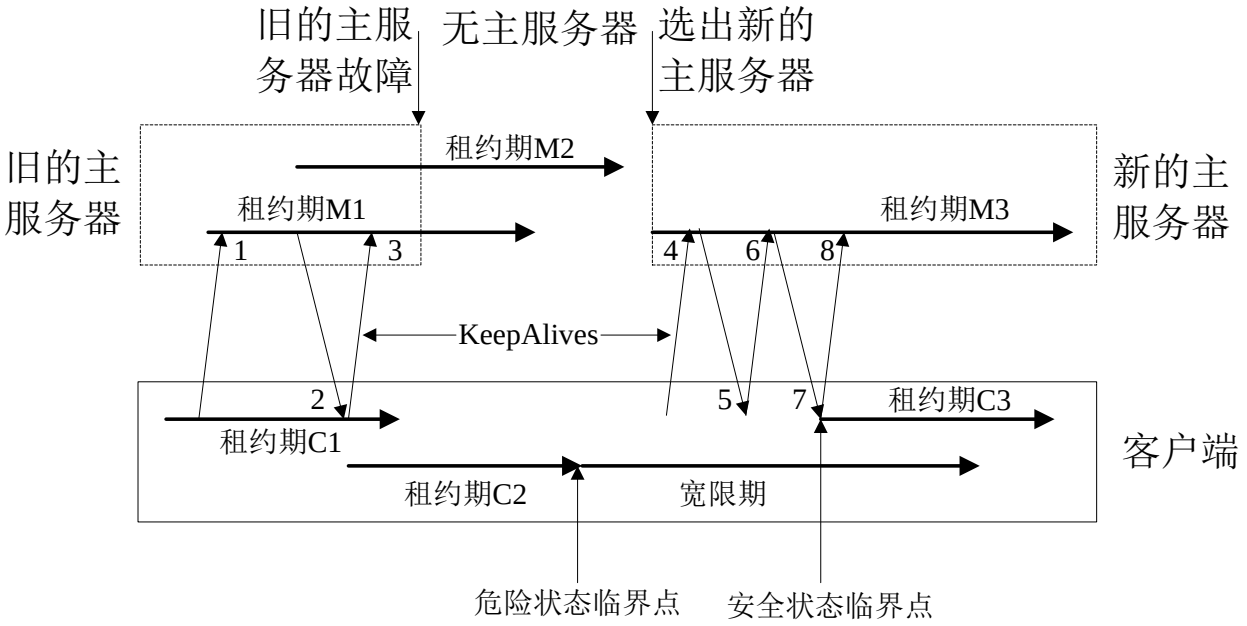
正常情况

- **Keep Alive是周期发送的一种信息。它主要有两方面的功能：延迟租约的有效期，同时携带事件信息告诉用户更新。主要的事件包括文件内容修改、子节点的增加、删除和修改等操作、主服务器出错等。正常情况下租约会由Keep Alive握手信息的重复而不断延长。**



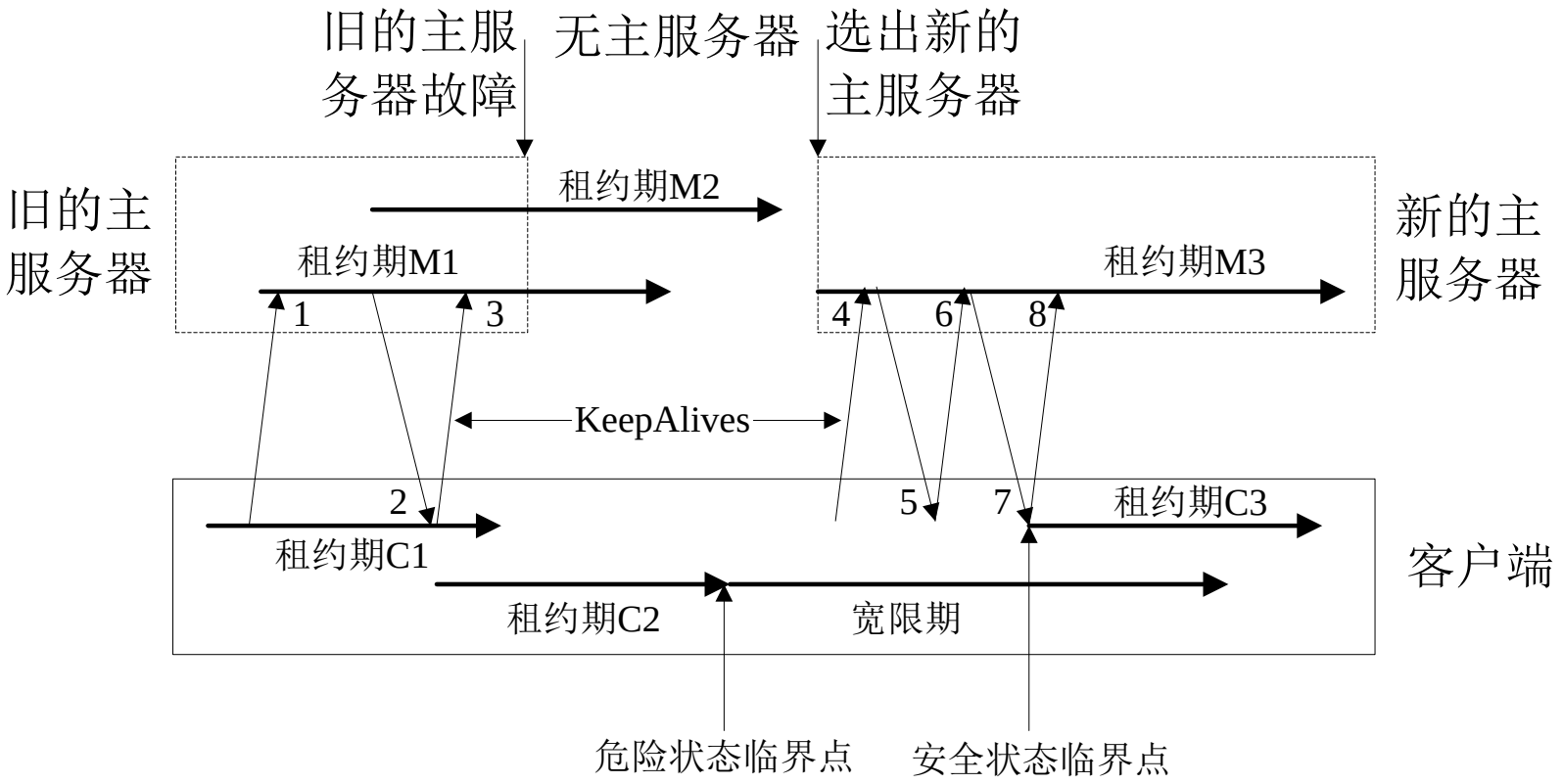
客户端租约过期

当客户端自身保存的租约过期，即没有收到主服务器的keep-alive回复，它将会进入“危险状态”。由于不确定会议是否被主服务器终止，它会Invalidate cache。同时，客户端进入寻找新Master的阶段。在这个阶段，客户端不停的轮询Chubby Cell中非主服务器的其它节点以获得新的chubby cell的视图。当客户端收到一个肯定的答复时，它会向新的主服务器发送keep-alive消息，并且告诉主服务器自己处在“危险状态”，和新的主服务器建立session。然后把cache中的handle发送给主服务器刷新。如果在一段时间后，默认为45s，客户端不能和新的视图建立session，那么客户端认为session失效，最终中止session。 在这段时间内客户端不能够更改其缓存的信息以保证数据一致性。



主服务器租约过期

如果主服务器出错，则说明有一段时间没有接收到来自客户端的**Keep Alive**的消息。则主服务器也会进入一段等待期，如果在此期间没有响应，则主服务器认为客户端失效，需要把客户端的获得锁以及打开的临时文件清理掉，这些都需要通知复本，来保持副本之间的一致性。如果主服务器收到客户端的**keep-alive**消息，则继续这个**session**。这样就涉及到悬挂句柄与会议维持的问题，讨论的是只要有悬挂句柄，**session**就永远维持。事实上，当租约过期关闭会议的时候可以有悬挂句柄，只是将悬挂句柄无效化即可。



主服务器出错

在选举新主服务器期间，各副本都只响应客户端的获得新视图的命令，而忽略其他命令。然后上任的新主服务器进行以下操作：

1) 选择新的编号，即新的试图编号，不接受旧主服务器的消息。

2) 只处理主服务器位置相关的信息，不处理session相关的信息。

3) 等待客户端的携带处于“危险状态”标志的keep-alive消息。

4) 主服务器响应客户端的keep alive，建立新的session，同时拒绝其他session相 关的操作。同时，向客户端返回keep-alive消息，警告客户端master fail-over，客户端需要更新handle和lock

5) 主服务器等待所有客户端的session确认keep-alive，或者让该session过期。如 果主服务器收到客户端的需要刷新的handle，则对这些handle进行更新，以区别于旧视图，并且在客户端的session以及自己的handle表中添加这些handle。

6) 主服务器可以响应客户端的所有操作。

7) 一段时间过后，主服务器开始检查是否有临时文件，以及是否存在一些lock没 有对应的打开的handle。如果临时文件或者lock没有对应的打开的handle，主服务器则删除临时文件、释放lock，这些也要保证新的视图的数据的一致性。

分布式锁服务Chubby的应用

- 主节点选举
- 独占锁
- 共享锁
- 数据存取应用
  - 获取GFS ChunkServer信息
  - 元数据存储
  - ……

24

Google文件系统GFS

- 系统架构
- 容错机制
- 系统管理技术

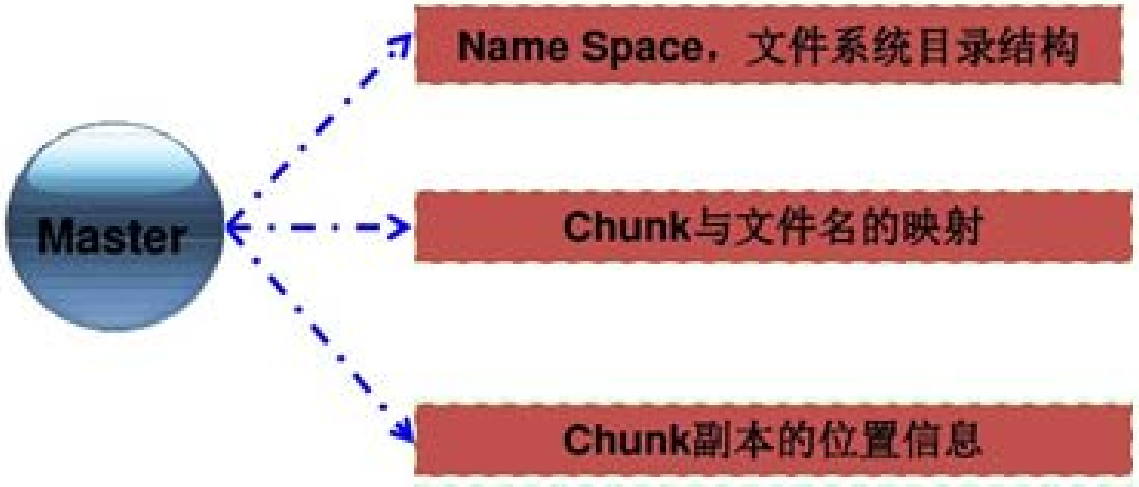
Master容错


Master
Name Space，文件系统目录结构
Chunk与文件名的映射
Chunk副本的位置信息(默认有三个副本)

单个Master，对于前两种元数据，GFS通过操作日志来提供容错功能

第三种元数据信息保存在各个Chunk Server上，Master故障时，磁盘恢复

GFS还提供了Master远程的实时备份，防止Master彻底死机的情况



Chunk Server容错

- 采用副本方式实现Chunk Server容错
  - 每一个Chunk有多个存储副本（默认为三个），分布存储在不同的Chunk Server上用户态的GFS不会影响Chunk Server的稳定性
  - 副本的分布策略需要考虑多种因素，如网络的拓扑、机架的分布、磁盘的利用率等
  - 对于每一个Chunk，必须将所有的副本全部写入成功，才视为成功写入

尽管一份数据需要存储三份，好像磁盘空间的利用率不高，但综合比[illegible]采用副本无疑是Simple, and good enough![illegible]度也最小的一种方法。

Google文件系统GFS

- 系统架构
- 容错机制
- 系统管理技术


GFS集群中通常有非常多的节点，需要相应的技术支撑
大规模集群安装技术
故障检测技术
GFS构建在不可靠廉价计算机之上的文件系统，由于节点数目众多，故障发生十分频繁
系统管理技术
新的Chunk Server加入时，只需裸机加入，大大减少GFS维护工作量
节点动态加入技术
节能技术
Google采用了多种机制降低服务器能耗，如采用蓄电池代替昂贵的UPS

GFS、HDFS等分布式文件系统对比

分布式文件系统很多：

- GFS　　HDFS
- 淘宝开源的TFS
- Facebook Haystack
- Tencent用于相册存储的TFS（Tencent FS，为了便于区别，后续称为QFS），

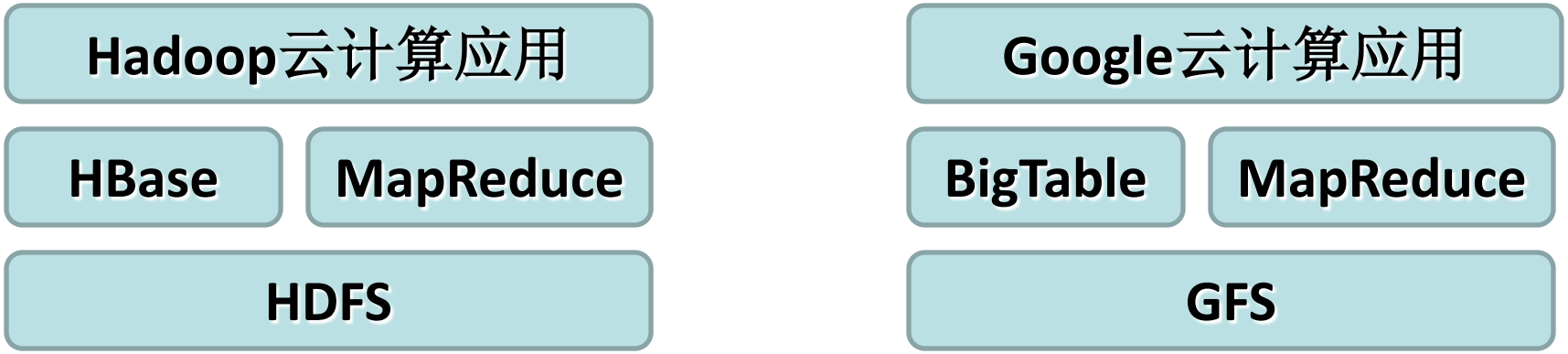
  TFS，QFS以及Haystack需要解决的问题以及架构都很类似，这三个文件系统称为Blob FS（Blob File System）。

HDFS是GFS的简化版，二者有很多相似之处

- **都采用单一主控机+多台工作机的模式，**由一台主控机(Master)存储系统全部元数据，并实现数据的分布、复制、备份决策，主控机还实现了元数据的checkpoint和操作日志记录及回放功能。工作机存储数据，并根据主控机的指令进行数据存储、数据迁移和数据计算等。
- **都通过数据分块和复制（多副本）来提供更高的可靠性和更高的性能。**当其中一个副本不可用时，系统都提供副本自动复制功能。同时，针对数据读多于写的特点，读服务被分配到多个副本所在机器，提供了系统的整体性能。
- **都提供了一个树结构的文件系统，**实现了类似与Linux下的文件复制、改名、移动、创建、删除操作以及简单的权限管理等。

GFS、HDFS总体架构比较

- 数据结构化管理组件：Hbase→BigTable
- 并行计算模型：MapReduce→MapReduce
- 分布式文件系统：HDFS→GFS
- Hadoop缺少分布式锁服务Chubby



GFS、HDFS子服务器管理模式差异

- **GFS：Chunk Server在Chubby中获取独占锁表示其生存状态，Master通过轮询这些独占锁获知Chunk Server的生存状态。GFS中，Master损坏时，替补服务器可以快速获知Chunk Server的状态。**
- **HDFS：DataNode通过心跳的方式告知NameNode其生存状态。HDFS中，NameNode损坏后，NameNode恢复时需要花费一段时间获知DataNode的状态**

在添加数据存储节点时，GFS的伸缩性较HDFS要好。

原因：Hadoop缺乏分布式锁服务

GFS、HDFS在关键点的设计上差异

- **GFS对多客户端并发追加同一个文件的多客户端并发Append模型**

GFS允许文件被多次或者多个客户端同时打开以追加数据。如果每次追加都访问GFS Master显然很低效，

GFS通过Lease机制将每个Chunk的写权限授权给Chunk Server。

写Lease的含义是Chunk Server对某个Chunk在Lease有效期内有写权限，拥有Lease的Chunk Server称为Primary Chunk Server，如果Primary Chunk Server宕机，Lease有效期过后Chunk的写Lease可以分配给其它Chunk Server。多客户端并发追加同一个文件导致Chunk Server需要对记录进行定序，客户端的写操作失败后可能重试，从而产生重复记录，再加上客户端API为异步模型，又产生了记录乱序问题。

Append模型下重复记录、乱序等问题加上Lease机制，尤其是同一个Chunk的Lease可能在Chunk Server之间迁移，极大地提高了系统设计和一致性模型的复杂度。

GFS、HDFS在关键点的设计上差异

- **HDFS文件只允许一次打开并追加数据，**客户端先把所有数据写入本地的临时文件中，等到数据量达到一个Chunk的大小（通常为64MB），请求HDFS Master分配工作机及Chunk编号，将一个Chunk的数据一次性写入HDFS文件。
- 由于累积64MB数据才进行实际写HDFS系统，对HDFS Master造成的压力不大，不需要类似GFS中的将写Lease授权给工作机的机制，且没有了重复记录和乱序的问题，简化了系统的设计。
- **HDFS由于不支持Append模型带来的很多问题，**构建于HDFS之上的Hypertable和HBase需要使用HDFS存放表格系统的操作日志，由于HDFS的客户端需要攒到64MB数据才一次性写入到HDFS中，Hypertable和HBase中的表格服务节点(对应Bigtable中的Tablet Server)如果宕机，部分操作日志没有写入HDFS，可能会丢数据。

GFS、HDFS在关键点的设计上差异

Master单点失效的处理。

- GFS中采用主从模式备份Master的系统元数据，当主Master失效时，可以通过分布式选举备机接替主Master继续对外提供服务，（Replication及主备切换本身有一定的复杂性）
- HDFS Master的持久化数据只写入到本机（可能写入多份存放到Master机器的多个磁盘中防止某个磁盘损害），出现故障时需要人工介入。

GFS、HDFS在关键点的设计上差异

对快照的支持

- GFS通过内部采用copy-on-write的数据结构实现集群快照功能
- HDFS不提供快照功能。

在大规模分布式系统中，程序有bug是很正常的情况，虽然大多数情况下可以修复bug，不过很难通过补偿操作将系统数据恢复到一致的状态，往往需要底层系统提供快照功能，将系统恢复到最近的某个一致状态。

HDFS基本可以认为是GFS的简化版，由于时间及应用场景等各方面的原因对GFS的功能做了一定的简化，大大降低了复杂度。

Blob File System 与 GFS/HDFS的差异

Blob File System的需求和GFS/HDFS其实是有区别的

（1）GFS和HDFS比较通用，可以在GFS和HDFS上搭建通用的表格系统，如Bigtable，Hypertable以及HBase，而Blog File System的应用场景一般为图片，相册这类的Blob数据。

（2）GFS的数据是一点一点追加写入到系统的，而Blob数据一般是整个Blob块一次性准备好写入到Blob文件系统，如用户上传一个图片。

（3）GFS是大文件系统，考虑吞吐量，可以在上面搭建通用KV或者通用表格系统，而Blob文件系统是小文件系统，一般只是用来存放Blob数据。

Blob File System 与 GFS/HDFS的差异

Blob File System的业务场景和GFS/HDFS其实是有区别的

在Blob FS中，由于整个Blob块数据一次准备就绪，Blob FS的数据写入模型天生就是比较简单，每次写入都请求Master分配Blob块编号及写入的机器列表，然后一次性写入到多台机器中。然而，**Blob FS面临的挑战是元数据过于庞大的问题**。由于Blob FS存储的Blob块个数一般很大，比如TFS中存储了百亿级的淘宝图片，假设每张图片的元数据占用20字节，所有的元数据占用空间为10G * 20 = 200GB，单机内存存放不下，并且数据量膨胀很快。因此，**TFS, QFS以及Facebook Haystack都采取了几乎相同的思路，Blob FS不存放元数据，元数据存放到外部的系统中。**

Blob File System 与 GFS/HDFS的差异

Blob File System的业务场景和GFS/HDFS其实是有区别的

比如，淘宝TFS中的元数据为图片的id，这些图片id存放在外部数据库，比如商品库中，外部数据库一般是Oracle或者Mysql sharding集群。

Blob FS内部也是按照Chunk块组织数据，每个Blob文件是一个逻辑文件，内部的Chunk块是一个物理文件，多个逻辑文件共享同一个物理文件，从而减少单个工作机的物理文件的个数。由于所有物理文件的元数据都可以存放到内存中，每次读取Blob逻辑文件只需要一次磁盘IO，基本可以认为达到了最优。

Blob File System 与 GFS/HDFS的相似

GFS与Blob FS看起来也有很多相似之处。

- 比如GFS和TFS目前都采用单一主控机+多台工作机的模式，主控机实现数据的分布、复制、备份决策，工作机存储数据，并根据主控机命令进行数据存储，迁移等.

分布式文件系统对比

- HDFS和GFS可以认为是类似的，GFS基本覆盖了HDFS的功能，而Blob FS和GFS面临的问题不同，设计的出发点也不一样，两类系统有本质的差别。
- 如果需要将GFS和Blob FS统一成一套系统，这套系统需要同时支持大文件和小文件，且这套系统因为存放的元数据量太大，Master节点本身也需要设计成分布式。这个大一统的系统实现复杂度非常高，目前只有GFS v2有可能可以做到。

总 结

HDFS, GFS, Blob FS

- 业务需求和设计目的
- 体系架构及其特点
- 具体市场应用场景和适用性
- 大数据分布式系统的未来发展趋势