Linkedin分布式对象存储系统Ambry

在Linkedin这样的社交网络中，全球各地大量用户上传与预览大规模的照片、视频以及文档文件等媒体对象（Media Objects）,存储这些媒体对象有如下挑战:

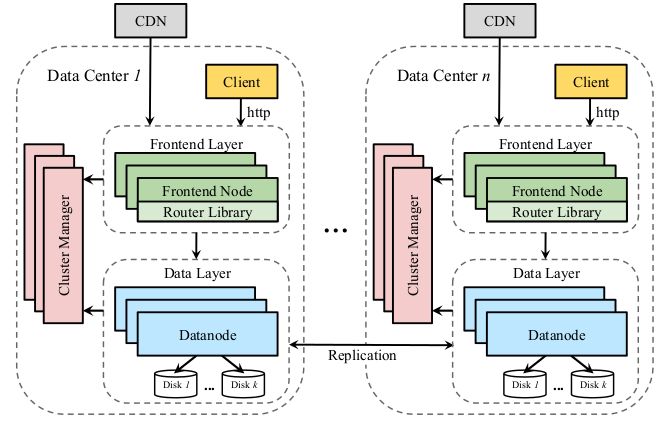
1. 这些媒体对象具有一次上传、不被更改以及极少删除的特点，且具有从KBytes到GBytes的不同大小分布
2. 需要对媒体对象快速且持久化存储，且具有高可用性
3. 每天用户数量、数据量与请求量的在快速增长
4. 由于不同媒体对新操作具有不同的工作负载，且集群扩展也会造成负载不均衡，因此需要负载均衡

为了解决这些问题，Linkedin开源了其分布式对象存储系统Ambry，其是一个不可变的对象存储系统，易于扩展，能够存储KB到GB大小的不可变对象，并且能够实现高吞吐和低延迟，该系统还支持跨数据中心的双活部署，并且存储成本低廉，适合于存储各种媒体内容，其具有的特点如下：

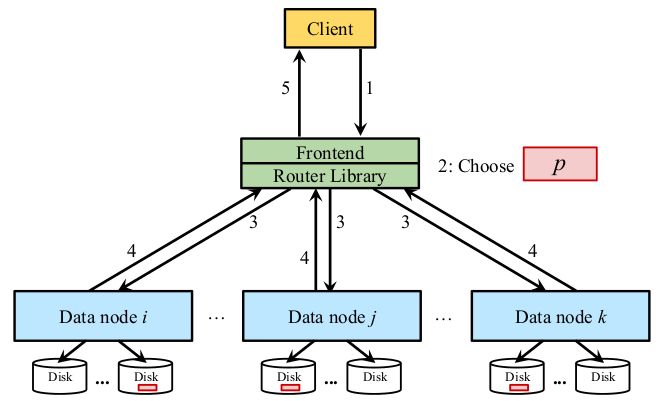
* 高可用性和水平可扩展，该系统要处理实时流量，会直接影响到站点的可用性，因此它必具有很高的可用性。另外，还希望能够尽可能的实现无缝的集群扩展
* 降低运维的负担，分布式系统一般很难管理，对于频繁的集群操作，能够实现自动化是非常重要的
* 更低的MTTR，分布式系统出现故障是难以避免的，但是很重要的一点在于快速恢复故障，让各个子组件启动并运行
* 跨DC双活，Linkedin有多个数据中心，因此所有的系统都要支持双活配置，这样系统就能够更新不同数据中心的同一个对象
* 提升小对象和大对象的效率，请求由小对象和大对象所组成的，小对象通常1K到100K，超出这个范围的对象会位于大对象桶（bucket）中。用同时处理各种大小的对象，通常来讲是很困难的。大量小对象会给元数据带来很高的负载，造成硬盘碎片，需要很多的随机IO，而大对象则需要很好的内存管理、端到端的流处理和有限的资源使用。
* 廉价，媒体内容很快占据大量存储空间，需要将旧数据变成冷数据，并不会频繁访问。针对这种情况，可以使用密集的硬件、分层存储、ErasureCode以及数据去重等。在设计时，Ambry希望媒体内容能够高效存储在密集型的机器上，并且能够非常容易的使用其他优化成本的方案。

# 1.概览

Ambry由三部分组成：数据节点（Storage Node），用来存储和检索数据；前端节点（Fronted Machines），将请求预处理后路由到数据节点；集群管理器（cluster manager），协调和维护集群。数据节点会在不同节点之间复制数据，同时支持数据中心内部和数据节点之间的复制。前端提供支持对象的PUT、GET和DELETE操作的HTTP API。另外前端所使用路由库也可以直接用在客户端中，从而实现更好的性能。在Linkedin，这些前端节点会作为CDN的原始服务器。系统结构图如下所示：



Ambry是基于句柄的存储系统，客户端将数据存储到Ambry中时会返回ID，其简化系统的设计而且数据存储去中心化。在ID中编码了获取Blob所需要的信息，这意味着写入Ambry的数据object不可变。对Ambry的访问流程如下：

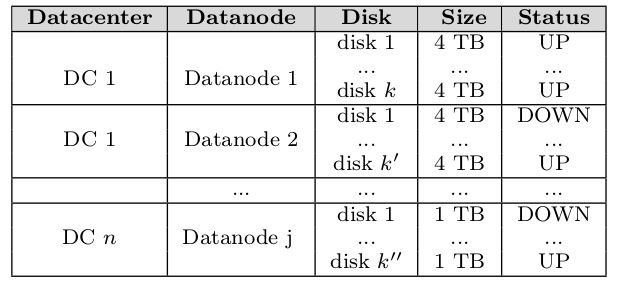


# 2. Ambry相关概念

Ambry使用ClusterMap维护集群的拓扑结构、状态及协调集群的操作，Clustermap由两部分组成Hardware Layout和Partition Layout。数据节点和前端节点都可以访问Cluster Map，并始终使用当前状态作出决策。

## 硬件布局（Hardware Layout）

Ambry集群使用异构的软硬件环境，因此使用硬件布局配置机器列表、每台机器上的硬盘以及每个硬盘上的容量。Layout还维护了资源（机器与磁盘）的状态（健康UP或者故障DOWN），并指定主机名与端口，通过其访问数据节点，如下表所示：

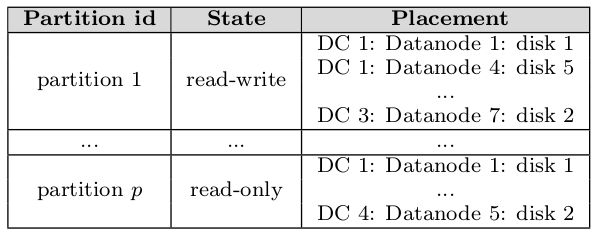


硬件布局配置在HardwareLayout.json文件中，集群管理器在启动时读取并加载该文件，加载后配置保持在ambry-fronted及ambry-server进程中，其格式如下：

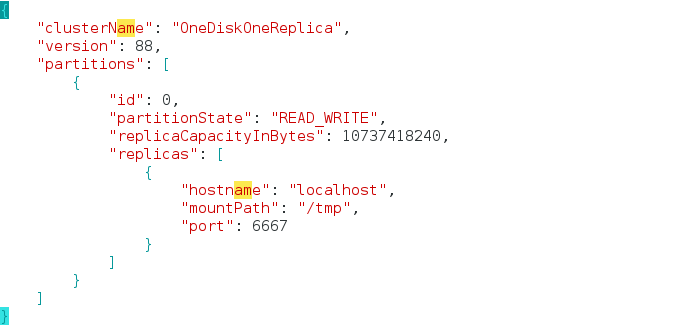


## 分区布局（Partition Layout）

分布布局包含了partition副本信息与状态（read-only或read-write）。集群管理器周期性的与数据中心交互更新partition状态信息。Partition的副本可以存在一个数据中心的多个数据节点或者不同数据中心中。Partition Layout的配置内容如下表所示：

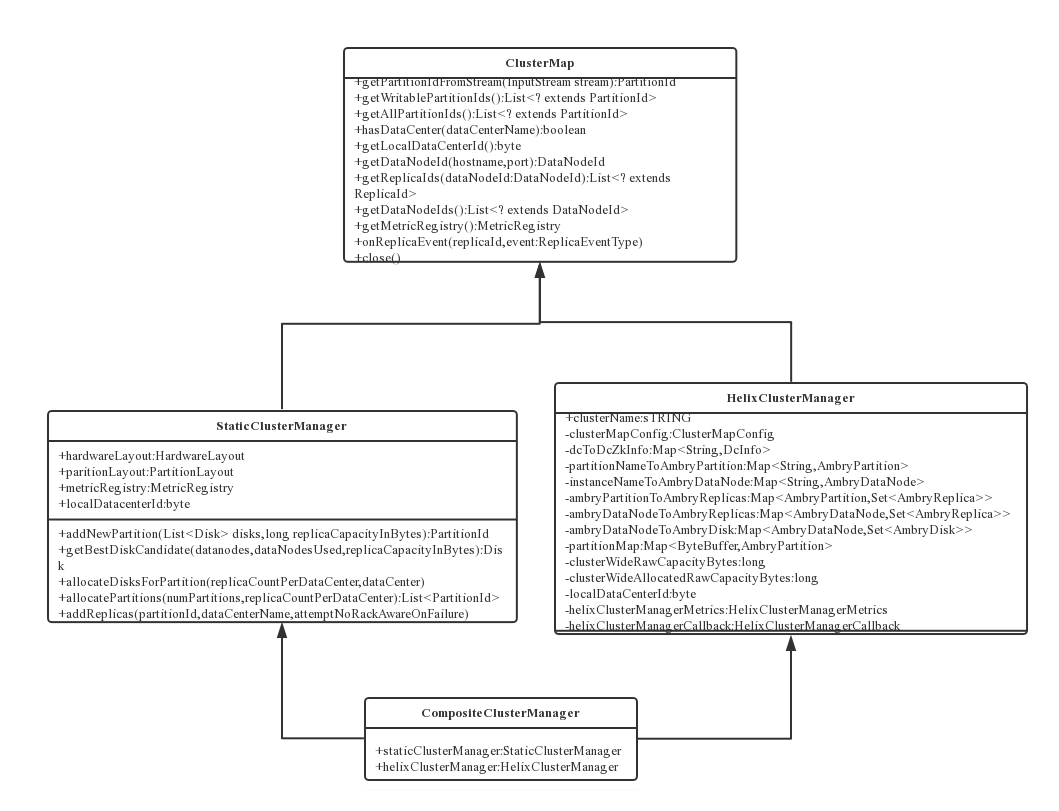


Partition Layout配置在PartitionLayout.json，如下所示：



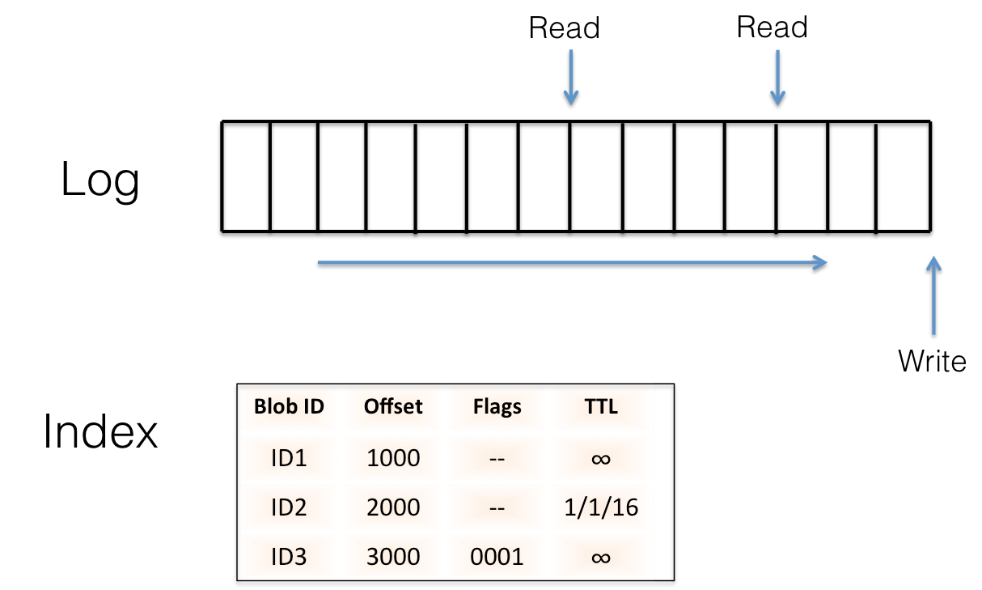
## Cluster Manager

CM从配置目录中读取HardwareLayout.json和PartitionLayout.json配置，基于这些配置进行Ambry集群的拓扑结构、状态及协调集群的操作。Cluster Manager的实现由多种：



## 存储

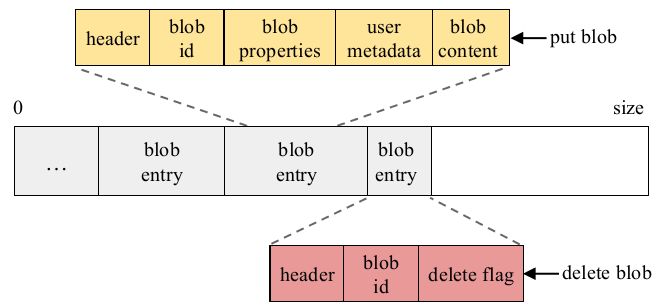
存储节点会用来存放不同分区的副本，每个存储节点会有N块磁盘，副本会跨磁盘分布存储，这些副本的结构和管理都是相同的，在数据节点上replicated store都会维护一个文件，这个文件称为on-disk log，在磁盘上会预先构建该文件，文件结构如下：



### 2.4.1 持久化(Log文件)

磁盘上每个副本均被建模为预先分配的log(preallocated log)，所有新的消息都会按照顺序附加到log上，消息是由实际的对象块（chunk）和相关的元数据（系统和用户）所组成。这能够使写入操作实现很高的吞吐量，并且避免出现磁盘碎片。Ambry会使用索引将对象id与log中的消息映射起来，索引本身是一组排序的文件片段，条目按照最新使用在前，最旧的条目在后排序，从而便于高效查找。索引中每个条目都会维护long中消息的偏移量、消息属性以及一些内部使用的域。索引中的每个片段会维护一个bloom filter，从而优化实际磁盘操作所耗费的时间。

随机选择blob对象组成逻辑单元组Partition，前端节点的路由库将逻辑分布与物理解耦的方式有利于再平衡时数据迁移透明，并且相比于Chord与CRUSH算法避免集群扩增时立即rehash的开销

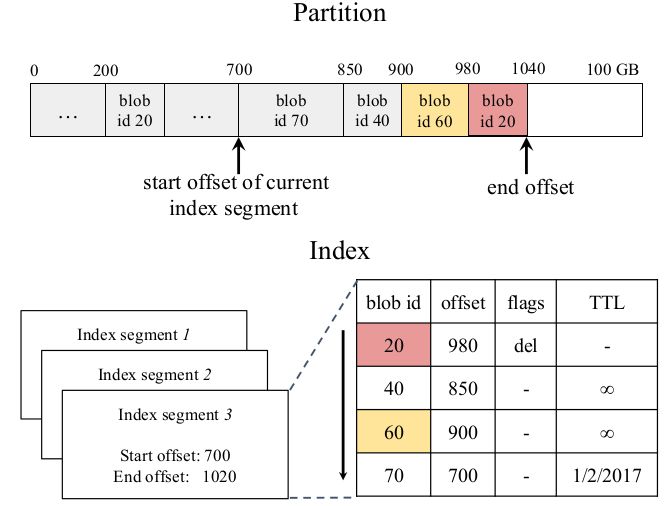


Partition是通过仅允许向后添加的预先分配的大日志实现，且是固定大小。对象被顺序写入到partition中作为put entry或者delete entry。对象id为对象唯一标志，由8字节的partitionid以及32字节的UUID组成，partitionId通过前端节点在put操作时为了数据平衡随机产生，且相同对象的多次put操作产生的id相同，在get或者delete操作是用来定位对象。由于对象不可修改，通过在partition中添加包含删除标志的delete entry用来实现软删除，删除的对象可以在周期性的压缩机制下被从partition上清理掉。

为了实现高可用性以及容错处理，partition被作为复制单位被复制到多个不同节点上，通过基于硬盘剩余空间的贪婪算法来决定副本分布。

### 2.4.2 Indexing索引对象(Segmentid)

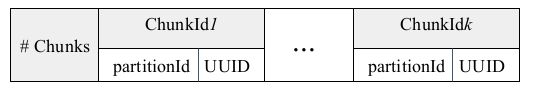
数据节点在内存中维护每个partition副本上对象偏移的索引数据结构，索引项由对象id排序，索引每项包含对象id到对象开始偏移的映射、删除标志以及TTL(time to live)，当Pub操作或者delete操作时均会更新index



与SSTable类似，Ambry限制了内存中index大小，将其切分成多个index segment，将旧的索引片段保持到硬盘上，并为每个硬盘上的索引片段维护一个内存上的Bloom Filter，用来快速过滤需要加载的硬盘索引片段降低硬盘检索延迟。索引片段按照逆时间顺序排序，这样查找一个对象时，如果对象被删除，其能够保证delete entry会比其他put entry先找到防止获取删除对象的数据。当数据中心故障，整个索引还可以根据partition重建。

### 2.4.3 分块技术（Chunking）

由于大对象会影响负载均衡，因此将大对象切分为多个固定大小的称为chunk的小对象，大小固定为4~8Mbytes，如下图：



在操作时，一个对象b会被顺序切分为k个chunk，每个chunk都视为一个独立的一般对象，均经过相同的put操作步骤加入到partition中，很大程度上会被放置在不同数据节点的partition上，因而可以并行写入。为了方便检索原始对象b而创建b的远数据对象bmetadata，其包含对象切分称的chunk数目以及顺序的ChunkId。

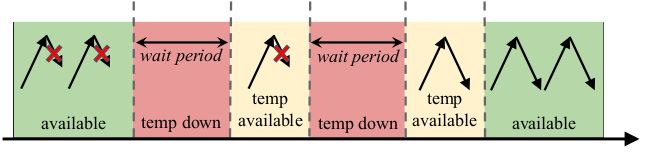
元数据bmetadata对象也被视为一般对象，在所有put操作完成后Ambry返回用户bmetadata的id来作为b的id。如果写入chunk操作失败，则会删除已经被写入的chunk并重新执行操作。在get操作时，对象的元数据对象bmetadata被检索出来，并从其中提取所有chunk的id。然后Ambry使用大小为s的滑动窗口缓冲区来获取对象数据，这是由于每个chunk可能分布在不同数据节点的partition上，因而可以并行地获取chunk数据。

当滑动窗口缓冲区中最开始的chunk已经被获取到，则缓冲区滑动一个chunk位置，并开始向用户返回对象数据。

### 2.4.4 其他

1）Zero-cost Failure Detection

大型系统中发生故障频率高，比如进程无响应、网络连接超时以及硬盘IO等问题，因此Ambry需要通过故障检测机制来发现服务的数据节点或者硬盘，并避免将请求转发给它们。在路由库中使用心跳与ping来实现零损失的故障检测，十分简单有效，消耗极少带宽。在不同状态对应的时间段内跟踪数据节点或者硬盘连续处理请求失败的次数，如果次数超过阈值则进行相应的状态转换



2）零拷贝，通过使用sendfile API，在进行读取时，字节从log转移到网络的过程中实现零拷贝。通过避免额外的系统调用，实现更好的性能，在这个过程中，会确保字节不会读取到用户内存中，不必进行缓存池的管理

3）恢复，因为系统和机器会出现宕机，磁盘上的数据由可能会损坏，所以哟必要实现恢复（recovery）的功能。在启动的时候，存储层会从最后一个已知的检查点读取log，并重建索引。恢复有助于重建内存中的状态，Log是恢复的来源，并且会永久保存

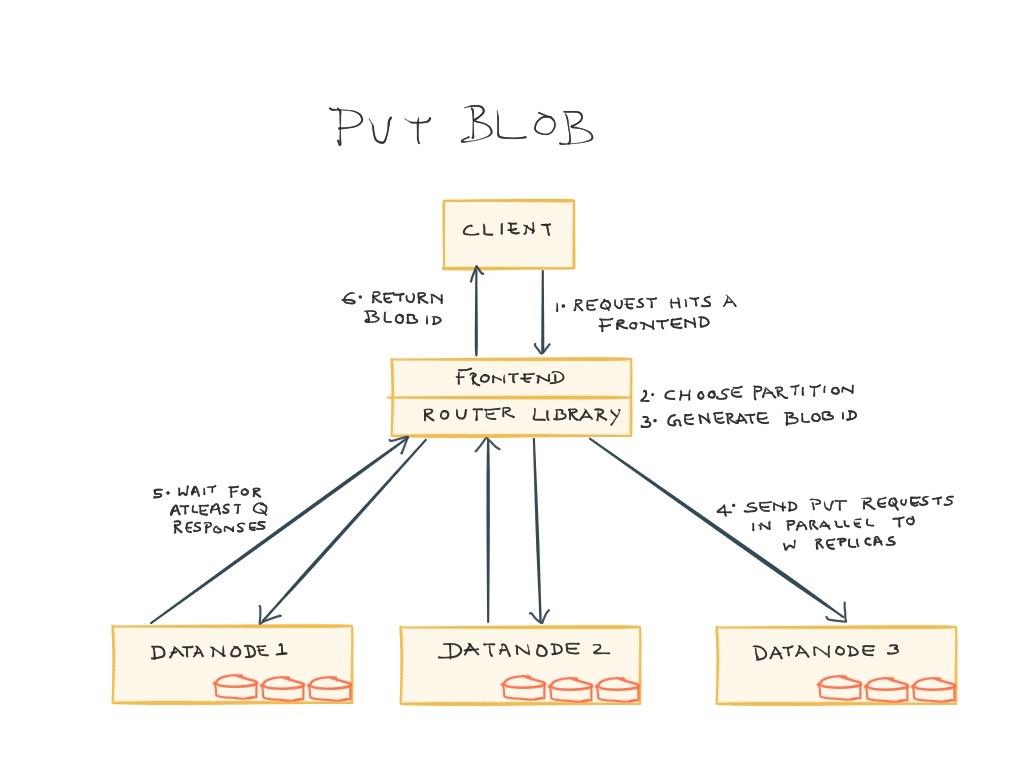
## 路由/前端

前端服务器提供了HTTP接口，供客户端与之通信，除此之外，还提供为CDN设置争取正确的头信息、进行安全校验，并对对象以流的形式返回给路由库和客户端。路由负责的功能如下所示：

1. 请求管理，请求的端到端生命周期是由路由来进行管理的，路由会处理PUT、GET以及DELETE请求。对于其中的每个请求类型，路由都会跟踪副本成功和失败的数量从而确定Quorum的值、维护分块的状态、生产对象id并在成功或失败的时候触发对应的回调。
2. 分块，大对象会分解称块，每个块都能够跨分区独立地进行路由。每个块都会由一个id来进行唯一标识。路由会生成一个元数据对象，其中包含了块的列表以及它们所需的获取顺序。元数据对象存储为独立的Blob，它的id也作为blob的id，在读取的时候，会得到元数据对象，然后检索各个块并返回给客户端。
3. 故障检测，负责主动识别宕机或状态出问题的资源，资源可以是机器、磁盘或分区，路由会将出现问题的资源标记为不可用，这样后续的请求就不会使用它们
4. Quorum，Ambry为写入和读取实现一种Multi-Master策略，这能够实现更高的可用性并减少端到端的延迟，这是通过减少一个额外的hop来实现，在基于主从结构的系统中，往往会有额外的hop。请求通常发往M个副本，然后等待至少N个成功的响应。路由会优先使用本地数据中心的副本，向其发送请求，如果本地存储无法实现所需的Quorum的话，它会代理远程数据中心的访问。
5. 变更捕获，在每次成功的PUT或者DELETE操作之后，路由会生产一个变更捕获（change capture）。变更捕获中所包含的信息是blob id以及blob相关的元数据，这个信息可以被下游应用所使用。

在路由中，典型的PUT操作和GET操作流程如下所示:

1. PUT Blob



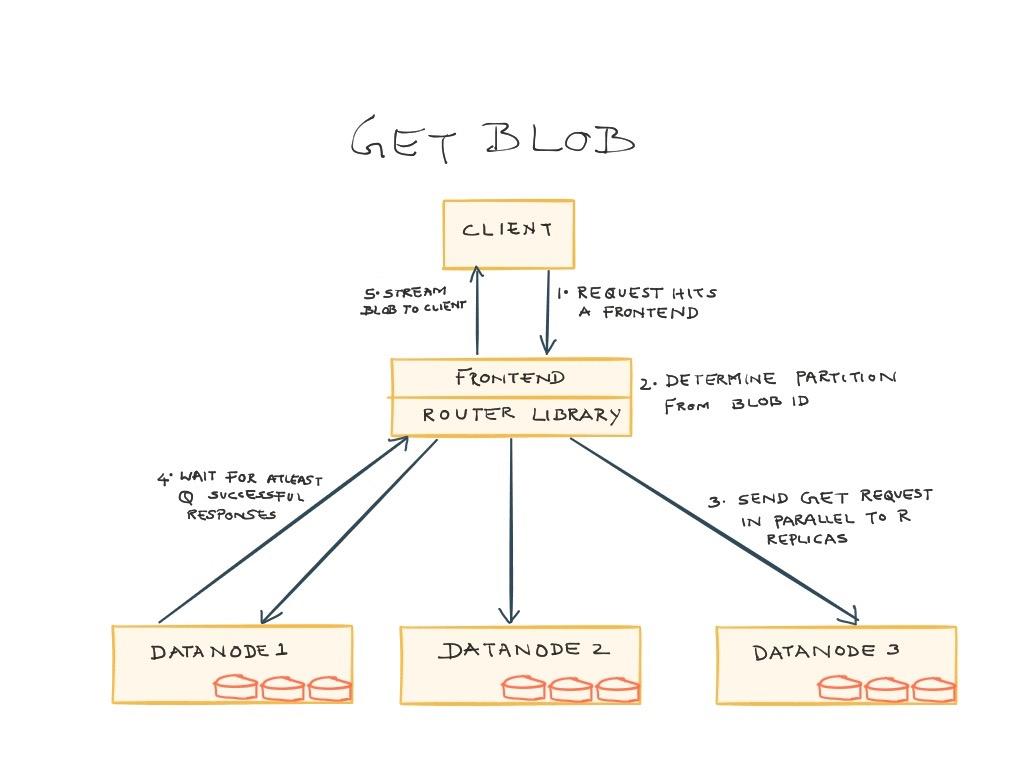
客户端会将对象以及一些元数据信息以流的形式发送到前端，当流到达时，前端会将对象进行分块、选择可用的分区、为blob或分块生成blob id，并将请求分发到W个副本。

然后，前端旧开始等待至少Q个成功的响应（Q<=W），等到之后，会将blob id返回给客户端。如果无法达到足够的Quorum，那么前端会报告一个错位。当然，Ambry也实现了当Quorum失败的时候，选择另外一个分区的功能，从而提升可用性。

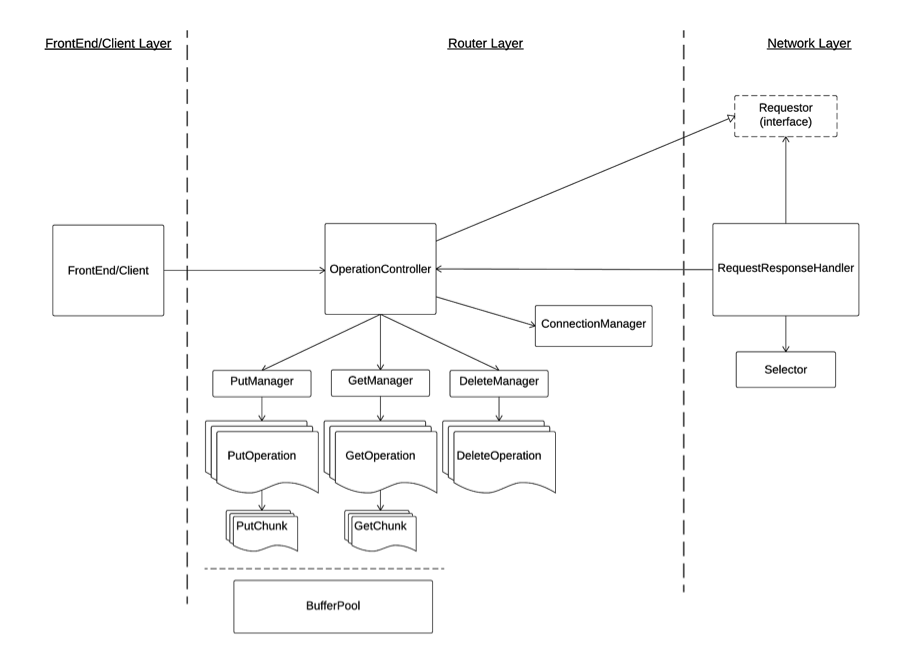
2）GET操作

客户端通过将id发送给前端来请求某一个Blob，前端会根据id来确定分区，并在数据节点中检索blob相关的块。

对于每个块，前端会并行发送R个请求，在将blob或分块发送给客户端之前，前端会等待Q个成功响应（Q<=R）。



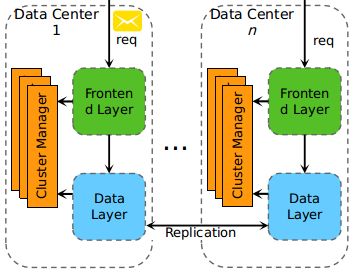
在Router中，请求的相关的类图如下：



* OperationController，Router包含了多个OperationController为格式的功能单元
* PutManager、GetManager、DeleteManager，每种操作都会对应一种Manager
* PutOperation、GetOperation和DeleteOperation，操作对应的对象
* PutChunk，每种操作包含固定数目的PutChunks
* PutMetadataChunk，包含多个metadata blob
* GetChunk
* ConnectionManager，维护与datanodes的连接
* RequestReponseHandler Thread，发送和接收NetworkSend和NetworkReceives
* BufferPool

## 复制

存储节点还需要维护分区中各副本的同步，每个节点上都会由一个复制服务（replication service），它会负责保证本地存储中的副本与所有的远程副本是同步的。在这里进行很多的优化，以保证复制过程的高效可靠，而且Ambry支持数据中心双活，数据能够被读取或者任意复制块中：



在AmbryServer中包含Replication Service，Replication Manager中启动Threads来进行数据的复制。

https://zhuanlan.zhihu.com/p/24273654

分布式安装：

https://www.2cto.com/kf/201611/563291.html

https://news.cnblogs.com/n/546553/

https://github.com/linkedin/ambry/wiki

http://www.infoq.com/cn/news/2016/06/LinkedIn-Ambry

https://github.com/linkedin/ambry