大数据的特征：大量、多样、快速、价值化

大数据价值链：生成→收集→存储→分析
生成：被动记录、主动生成、自动产生
收集：Pull&Push、传输、预处理
存储：存储设施、数据管理、编程模型
分析：描述、预测、统计、挖掘、聚类

大数据问题：遍历、提取、聚合、生成
挑战：并行、分布式存储、节点故障管理、网络瓶颈

商用集群架构：16-64 节点/机架、1Gbps/架内节点速度、2-10Gbps/架间骨干网速

扩展问题 Scalability：增量数据、用户、问题的复杂性
2 种方法：纵向扩展（Scale up）、横向扩展（Scale out）
纵向：增强单节点的 CPU、内存性能
横向：增加节点

集群挑战：
分布式编程：问题分解与并行、任务同步
任务安排：提升速度、分配资源、减轻故障影响

IPS=(MF * IPC)/(F + (1-F)/N)
IPS 每秒处理指令数 MF 主频 IPC 每时钟周期处理指令
F 序列化部分 (1-F)可并行部分 N 处理器数量

Availability = 100% * MTTF / (MTTF+MTTR)
MTTF 平均无故障时间 MTTR 平均故障恢复时间

容错：负载均衡、容器隔绝、备份恢复

提速比例：$T_1/(N \cdot T_N) < 1$
效率：1/(F + (1-F)/N)

Lambda 架构：提供大数据的流式处理和批处理的方法
要求：容错（硬件、人为）、线性扩展能力、可拓展、多样任务（低延迟查询、更新等）
关于查询的指标：延迟、及时性（一致）、准确性
数据通道：冷通道(Batch 层)、热通道(Speed&Serving 层)
冷通道：批量处理新数据、提供 Batch view
热通道：实时分析数据，牺牲准确性换低延迟

Hadoop：开源可扩展的分布式计算软件平台
提供基于 MapReduce 的 API
提供任务分配、网页监视、故障处理、分布式文件系统
应用场景：可并行任务、可批量处理、大量非结构化数据
不应使用：高强度计算、不可并行、数据不自洽、需互动
应用举例：日志索引、数据排序、图像分析、搜索引擎优化

CAP 一致、可用 Partition-tolerance 分区容错
强一致性：顺序一致；弱一致性：最终一致

乐观锁：假定冲突会发生，禁止读取
悲观锁：假定冲突不会发生，更新后再检查冲突、回滚

Hadoop 对 CAP 的取舍：AP 与最终一致性

HDFS:

```
Client---Metadata ops.→NameNode
↑                              /
 --R/W→DataNode←Block ops.---
```

数据块 Block 通常 64-256MB
增大块的大小：计算效率低
减小块的大小：降低获取效率、浪费计算资源

NameNode 维护 HDFS 的元数据
FsImage 维护文件树和文件夹的元数据
文件：备份数、修改/访问时间、权限、数据块及大小
文件夹：修改时间、权限、配额
不记录数据和 Datanode 的关系，此映射由内存维护
数据块信息和记录由 DataNode 主动定时发送
Editlog 记录文件的修改操作
合并 FsImage 和 Editlog（基于 HTTP GET/POST）
将 FsImage 和 Editlog 发送至 SecondaryNamenode
清空 Editlog
SecondaryNamenode 合并为新的 FsImage
新的 FsImage 传回 Primary Namenode

命名空间：由文件夹、文件、数据块组成
HDFS 继承了传统的层级文件系统的设计，其增删改查操作均保持了传统。
命名空间保存在 Namenode 的内存中，单命名空间能够管理的文件数和数据块数受内存大小限制。
单命名空间存在受限于 Namenode 性能的瓶颈。
单命名空间无法隔绝集群中不同的应用。
单命名空间存在单点故障风险。

解决：HDFS 高可用（HA)/ HDFS 联邦（Federation）

底层通信协议：TCP/IP
客户端-Namenode：Client Protocol
客户端-Datanode: RPC 远程过程调用协议
Namenode-Datanode: Datanode Protocol

HDFS 数据存储
冗余：加速传输、检查错误、保证可靠
第一备份：保存在最初上传的 Datanode 中
第二备份：保存在与第一备份不同的机架上的随机节点
第三备份：保存在与第一备份相同的机架上的不同节点
更多备份：保存在随机节点
Network Traffic = Datasize*dfs.replication = Disk Space
数据读取：首选 Rack ID 相同的节点

数据错误与恢复
Namenode 故障：使用 SecondaryNamenode 的备份恢复
Datanode 故障：采用主动心跳机制，离线的 Datanode 会被标记为不可读。其中的数据会再次被备份。
每 10 次心跳附带 1 次数据块报告。
客户端读数据时，采用 MD5 等校验和检验数据块的正确性。

MapReduce
Map：将输入映射为中间数据表示
Reduce：将中间数据表示变成最终输出

#Bucket = hash(key) % #ReduceTasks 对相同的 key 聚类

Shuffle in Map Phase
1. Map output：生成 Map 后的键值对
2. Flush to disk
3. Spilling
4. Partitioning and Sorting：根据哈希划分记录，按键排序
5. Combiner：将具有相同键的记录合并（可选）类似 reduce
并非所有操作都可以作为 combiner
需要不依赖状态、具有交换律和结合律、与 reducer 一致
6. Merge
<k1, v1>, ..., <k1, vn> -> <k1, <v1, ..., vn> >

Shuffle in Reduce Phase
1. Fetch：Reducer 通过 RPC 请求 JobTracker
当 Map 结束时，JobTracker 会唤起 Reducer 去抓取 Mapper 机器上的 Spilling File
2. Merge
<k1, <v1, ..., vn> > -> <k1, V1>

To be implemented:
map(k1, v1) -> list(k2, v2)
reduce(k2, list(v2)) -> list(k3, v3)

Hadoop Architecture
Client->JobTracker->>TaskTrackers-->>DataNodes
_>NameNode
作业 Job：MapReduce 程序的完整计算过程
任务 Task：MapReduce 并行计算的基本任务
JobTracker 是向 Hadoop 提交、追踪 MapReduce 任务的守护服务。它接收客户端提交的 MapReduce 作业，与 Namenode 通信获得数据的位置，定位可用的 TaskTracker 节点，并向选定的 TaskTracker 节点提交任务。

TaskTracker 接收来自 JobTracker 的 Map，Reduce 或 Shuffle 操作。其需要向 JobTracker 发送心跳，并附加告知可用的计算单元数。

MapReduce 设计模式
-将问题转化为 过滤 或 聚合 的若干步骤
-MapReduce 适用于①很少更新的大文件②需要遍历整个文件，从中提取性质的情景。

Summerization Pattern:
计数、翻转索引、计数等
Filter Pattern:
筛选：<k1, v1>, <k2, v2> → <k1, v1> （仅 Map）
前 K：<k1, v1> → <局部前 K 记录> → <前 K 记录>
去重：<k1, v1> → <局部去重记录> → <去重记录>
Data Organization Pattern:
Binning：基于 Map-only 作业，将大数据集分类存放
Shuffling：打乱记录顺序
<offset, rec1> → <random k1, rec1> → <rec1, null>

MetaPattern：组织复杂应用中多个作业的工作流程
JobChain：一系列串联的作业
Join Pattern：实现关系代数中的交运算
Reduce-side Natural Join:
<offset, rec A1> → <外键, A + rec A1> → <null, A1 join B1>
<offset, rec B1> → <外键, B + rec B1> ... <null, Ai join Bj>
Map-side Natural Join：将小表作为分布式缓存，省去 reduce

Distributed Cache：分享、缓存只读的小文件以提升效率

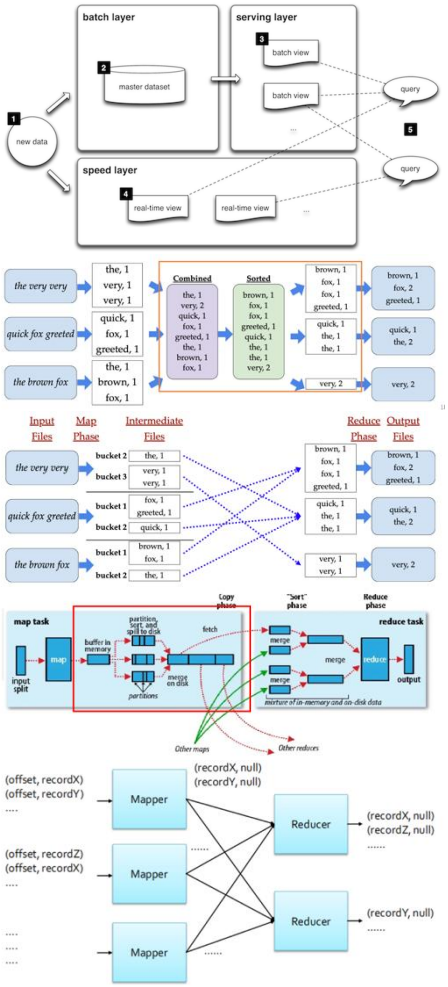

图 6：去重模式的结构