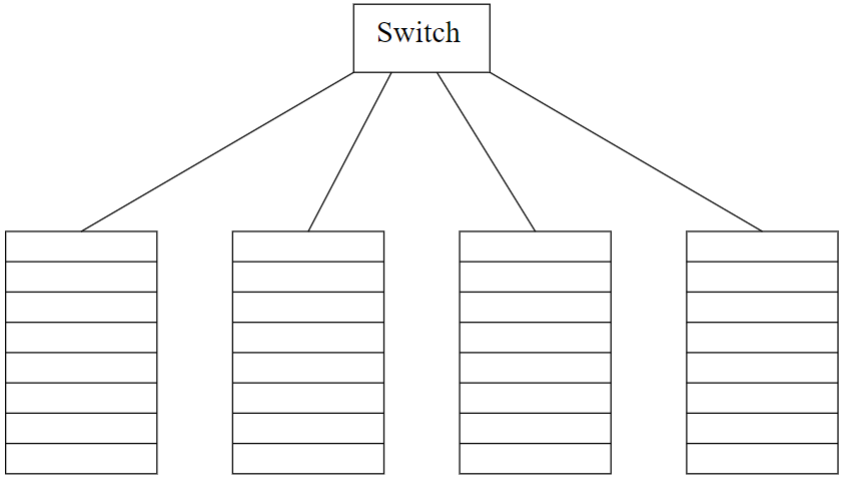
分布式文件系统

- 物理结构

- 分布式文件系统（DFS）

分布式文件系统——物理结构

- 结构:



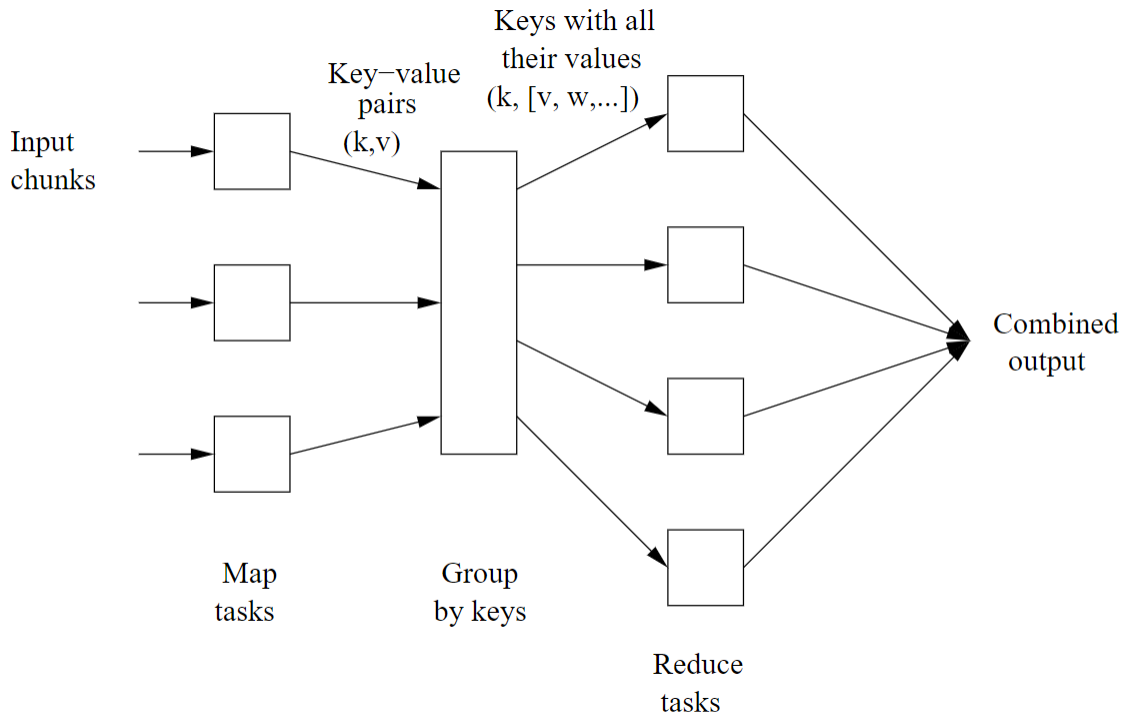
- 特征
  - 文件多副本存储
  - 计算过程分成多个任务

分布式文件系统——DFS

- 文件类型:
  - 文件非常大
  - 文件极少更新
- 文件被分成文件块（chunk），文件块被复制成多份放在不同机架的节点上
- 总目录用于定位主节点及其副本，主节点用于定位某个文件的文件块

MapReduce

- 过程：
  - Map任务：输入为文件块，输出为键值对序列。
  - 主控制器：将Map输出的键值对序列，按键分配到Reduce任务中。
  - Reduce任务：作用于一个键，将与此键关联的所有值组合起来，组合方式由Reduce函数决定。



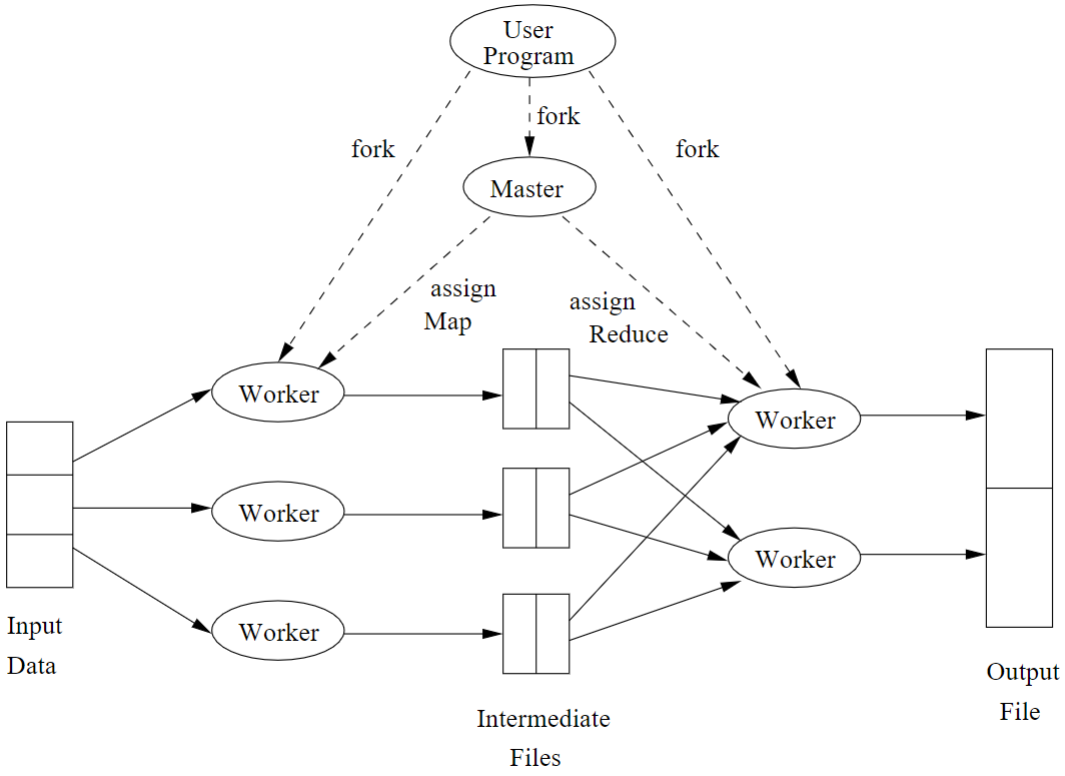
MapReduce——组合器

- 在Map任务产生键值对序列后，在主控制器执行分组和聚合的过程前，由Map任务应用Reduce函数，此过程为组合器。

- 每个Map任务在经过组合器后仍旧只会给出一个包含w的键值对，因此仍需执行分组和聚合后将结果作为Reduce的输入。

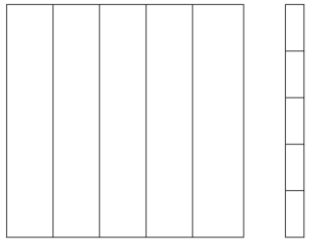
MapReduce——执行细节

- 用户程序fork一个主控进程和若干工作进程（工作进程分为处理Map任务的和处理Reduce任务的）
- 主控进程：
  - 创建Map任务和Reduce任务
  - 记录每个Map任务和Reduce任务的运行状态



基于MapReduce的矩阵-向量乘法

- 适用条件：有一个n很大的n×n的矩阵M，以及一个可以放入内存中的向量v
- Map：首先计算节点读入向量v；然后将v和M的一个文件块作为输入，对M中的每个元素m_{ij}，Map输出键值对（i，$m_{ij}v_j$）
- Reduce：将与键i关联的值相加得到结果（i，x_i）
- 若向量v很大，（即n很大），则将矩阵分割成多个宽度相等的垂直条，将向量分割成同样数目的水平条。第i个垂直条只和第i个水平条相乘。这样就可以把矩阵的每个条存成一个文件，向量的每个水平条存成一个文件。



基于MapReduce的选择运算

- Map：输入为一个元组t，检查t是否满足条件C，满足则输出一个键值对（t，t）

- Reduce：仅仅将键值对传递到输出部分。

基于MapReduce的投影运算

- Map：对元组t，剔除t中不在S中的属性字段，得到元组t‘，输出键值对（t’，t’）

- Reduce：剔除冗余元组，即存在一个或多个键值对（t’，t’），将（t‘，[t’,t’,…,t’]）转换成（t’，t’）

基于MapReduce的并运算

- Map：将输入元组t变为键值对（t，t）

- Reduce：对于每个键t，值表中可能有一个或两个值，即[t]或[t，t]，去除冗余，输出（t，t）

基于MapReduce的交运算

- Map：将输入元组t变为键值对（t，t）
- Reduce：如果键t的值表为[t，t]，则输出（t，t），否则不产生任何结果

基于MapReduce的差运算

- Map：对于关系R和S，对于R中的元组t，生成键值对（t，R）；对于S中的元组t，生成键值对（t，S）

- Reduce：若一个键t的值表为[R]，则输出（t，t），否则不产生任何结果。

基于MapReduce的自然连接运算

- Map：对于R中的每个元组（a，b），生成键值对（b，（R，a）），对S中的每个元组（b，c），生成键值对（b，（S，c））

- Reduce：对于每个键b，其值表为（R，a）或（S，c），生成三元组（a，b，c）

基于MapReduce的分组和聚合运算

- Map：对于每个元组（a，b，c），生成键值对（a，b）

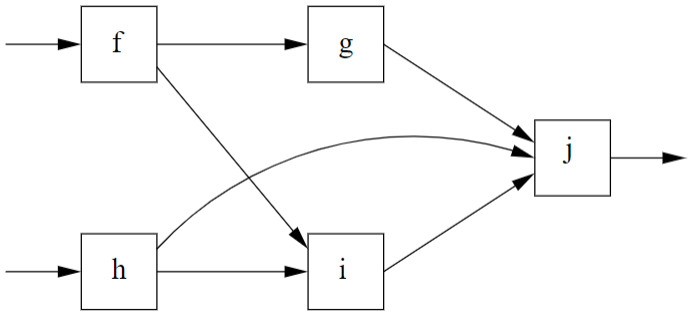
- Reduce：对于每个键a，与其关联的值表为[b_1,b_2,…,b_n]，对其施加运算（如SUM, MAX等），得到结果x，输出结果为（a，x）

基于MapReduce的矩阵乘法

- 第一步：
  - Map：对M中的矩阵元素m_{ij}产生键值对$(j, (M, i, m_{ij}))$，对N中的矩阵元素n_{jk}产生键值对$(j, (N, k, n_{jk}))$。
  - Reduce：对每个键j，其关联值表为(M, i, m_{ij})和(N, k, n_{jk})，产生键值对，其中键为（i，k），值为$m_{ij}n_{jk}$
- 第二步：
  - Map：恒等函数
  - Reduce：对每个键（i，k），计算此键关联的所有值的和，结果为（（i，k），v），v为矩阵P=MN第i行第k列的元素值

MapReduce的扩展——工作流系统

- 将MapReduce从一个简单的两部工作流扩展为函数集的任意组合，可以通过一个无环图来表示。
- 工作流中的每个函数可以由很多任务执行，由主控进程负责分割整个工作。
- 阻塞性：只有任务完成才会将输出传递给接收任务。保证任务失败后，主控进程能在其他计算节点上重启该任务，并且不会与之前传递给其他任务的输出重复。



MapReduce的扩展——Spark

- 引入弹性分布式数据集（RDD）
- “分布式”：一个RDD通常会被分解为多个块
- “弹性”：能从RDD的任意块的故障中回复
- “转换”操作：将一些函数应用于RDD生成另一个RDD
- “执行”操作：从周围的文件系统中获取数据，并产生一个结果返回给调用Spark的应用。

MapReduce的扩展——Spark的常用操作

- Map转换：可以应用于任意对象类型，但结果只会生成一个对象。
- Flatmap：不要求所有类型都是键值对，可以生成一组对象。
- Filter：采用谓词作为参数，谓词为每个对象返回true或false，输出仅由返回true的对象组成。
- Reduce：参数为一个函数，对于接收到的RDD，反复应用该函数到每队连续元素，并输出单个元素。

MapReduce的扩展——Spark实现

- 惰性评估：当在一个节点上创建一个RDD的分割块时，可以立即在同一计算节点上使用它对其应用另一个转换操作。这样，RDD就可以不存储在磁盘上，也不传输到另一个计算节点，能以数量级方式节省运行时间。
- 血统：替代冗余存储，记录创建的每个RDD的“血统”，告诉Spark系统如何重建该RDD或者在需要时重建RDD的分割块。

MapReduce的扩展——TensorFlow

- 使用张量来代替RDD

- TensorFlow程序无循环，而Spark可以进行代码段的反复迭代。

通信开销模型

- 如何度量MapReduce算法质量?

Communication cost = input file size + 2 × (sum of the sizes of all files passed from Map processes to Reduce processes) + the sum of the output sizes of the Reduce processes.

- eg：连接算法的通信开销

R（A,B）⋈ S（B,C），关系R和S的规模分别是r和s

- Map：r+s
- Map→Reduce：O(r+s)
- Reduce：O(r+s)，若输出规模很大，可通过聚合操作来减少输出规模

总：O(r+s)

通信开销模型

■ 多路连接的选择

R（A,B）⋈ S（B,C）⋈ T（C,D），关系R、S、T的规模分别是r、s、t，假定B、C字段一致概率为p

- 两次二路连接

按RST顺序连接，RS连接开销为O(r+s)，产生结果规模prs，再与T连接，通信开销为O(t+prs)，总通信开销为O(r+s+t+prs)，同理按STR顺序连接总通信开销为O(r+s+t+pst)

- 一次三路连接

计划k个Reducer，b、c代表将字段B、C哈希到的桶数目，h、g分别表示其哈希函数，要求**bc=k**，即每个Reducer对应**一对桶**，当h(v)=i且g(w)=j时，对应桶对(i,j)的Reducer就负责连接R(u,v)，S(v,w)，T(w,x)

总通信开销为：

（1）**s** 将每个元组S(v,w)仅仅传递一次到Reducer(h(v),g(w));

（2）**cr** 将每个元组R(u,v)传递到c个Reducer(h(v),y)，y的可能取值有c个;

（3）**bt** 将每个元组T(w,x)传递到b个Reducer(z,g(w))，z的可能取值有b个;

（4）此外，将每个关系输入Map任务还有r+s+t的固定开销。

通信开销模型

- 多路连接的选择

问题转化为选择b、c，满足bc=k，使得s+cr+bt最小

采用拉格朗日乘子法，令s+cr+br-λ(bc-k)对b和c的偏导为0

当$c=\sqrt{kt/r}$， $b=\sqrt{kr/t}$时通信开销取最小值$r+2s+t+2\sqrt{krt}$

MapReduce复杂性理论

■ MapReduce算法族参数

- Reducer规模：单个键的关联值表中元素数目的上界
- 复制率：所有Map任务在所有输入上产生的键值对的数目除以输入的数目，即Map任务到Reduce任务每个输入上的平均通信开销

■ eg：相似性连接

假定有一个100万张图片组成的数据集，每张图片1MB，给定相似度度量函数s(x,y)，寻找相似图片对。MapReduce：输入为键值对(i,P_i),i为图片ID，P_i为图片本身，对每对图片进行对比，对应Reducer输入为$(\{i,j\},[P_i,P_j]\})$。该方法Reducer规模极小，但复制率为999999，因此Map与Reduce之间的通信总量为1000000（图片总数）*999999（复制率）*1000000（每张图片大小）。实际应考虑Reducer规模与复制率的折中，既保证函数需要的内存不超过可用内存，又尽可能降低通信开销。

MapReduce复杂性理论

■ eg：相似性连接

通过将图片分为m组（每组$10^6/m$张）：

Map函数：对输入元素(i,P_i)生成m-1个键值对，键是可能的集合{u,v}中的一个，其中u是图片i所在的组，v是另一个组，关联的值为(i,P_i)。

Reduce函数：考虑键{u,v},关联值列表包含$2*10^6/m$个元素，对列表中不同组的图片应用相似度函数s。此外，在编号为{u,u+1}的Reducer上比较u组的元素。

复制率：m-1；Reducer规模：$2*10^6/m$

■ 映射模式

- 映射模式定义为表示算法所用的多个Reducer产生输出的过程，即对于给定问题给定Reducer规模q时，映射模式指输入到一个或多个Reducer的分配方式。需满足：

（1）任何一个Reducer都不能分配超过q个输入；

（2）对于问题每一个输出，至少有一个Reducer会被分配与输出关联的所有输入。此时称该Reducer覆盖（cover）了输出。

MapReduce复杂性理论

■ 复制率的下界

如何知道获得了最优的折中结果？只能知道最小可能的通信量。

求复制率的下界的一般步骤：

（1）对于有q个输入的Reducer能够覆盖的输出数目，给出一个上界，记为g(q);

（2）确定问题产生的输出总数；

（3）假设有k个Reducer，其中第i个Reducer有q_i<q个输入，$\sum_{i=1}^{k} g(q_i)$一定不会小于输出总数；

（4）对（3）得到的不等式进行处理，会得到$\sum_{i=1}^{k} q_i$的一个下界，通常使用的技巧为q_i替换为其上界q；

（5）由于$\sum_{i=1}^{k} q_i$是从Map任务到Reduce任务的通信总量，将（4）中得到的下界除以输入的数目，即为复制率的下界。

■ eg：“所有对”问题复制率的下界

（1）q为Reducer规模，则Reducer最多得到q个输入，其覆盖的输出数目不可能超过$C_q^2 \approx {}^1/_2\, q^2$;

（2）p为输入数目，则输出总数最多为$C_p^2 \approx {}^1/_2\, p^2$；

MapReduce复杂性理论

- eg：“所有对”问题复制率的下界

（3）构造的不等式为$\sum_{i=1}^{k}\frac{1}{2}q_i^2\geq\frac{1}{2}p^2$；

（4）两边乘以2并将一个q_i替换为q，由于q_i<q，不等式仍然成立：$q\sum_{i=1}^{k}q_i\geq p^2$；

（5）两边同时除以q以及输入的数目p，得到复制率的下界p/q。