Hadoop map/reduce 开发

Map/Reduce开发流程

- 新旧Map/Reduce框架
  - org.apache.hadoop.mapred
  - org.apache.hadoop.mapreduce
- 区别
  - JobConf和Job用于提交作业
  - Mapper/Reducer初始化函数Configure和Setup
  - OutputCollector<K,V>和Context
  - Counter和Reporter变化

Mapper开发

- class XX extends Mapper<K_in, V_in, K_out, V_out>
- Mapper设计
  - void setup(Context context)
    - context.getConfiguration();
  - void map(KEYIN key, VALUEIN value, Context context)
  - void cleanup(Context context)

Reducer开发

- class XX extends Reducer<K_in, V_in, K_out, V_out>
- Reducer设计
  - void setup(Context context)
    - context.getConfiguration();
  - void reduce(KEYIN key, VALUEIN value, Context context)
  - void cleanup(Context context)

ToolRunner

- extends Configured implements Tool
- int run(String[] args) throws Exception
- Main函数调用ToolRunner
  - ToolRunner.*run(new Configuration(), new CommonSort(),args);*

Map/reduce submit

```
Job job=new Job(new Configuration(),"CommonJoinx");
job.setJarByClass(CommonJoin.class);
job.setMapperClass(CommonJoinMapper.class);
job.setMapOutputKeyClass(TextPair.class);
job.setMapOutputValueClass(Text.class);
job.setCombinerClass(CommonJoinCombine.class);
//设置key分区
job.setPartitionerClass(CommonJoinPartitioner.class);
//设置分组
job.setGroupingComparatorClass(CommonJoinComparator.class);

job.setReducerClass(CommonJoinReducer.class);
job.setOutputKeyClass(IntWritable.class);
job.setOutputValueClass(Text.class);

job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);

TextInputFormat.setInputPaths(job, new Path(args[0]));
TextOutputFormat.setOutputPath(job, new Path(args[1]));
job.getConfiguration().set("mapred.mapper.left.table","/movie_genre.txt");
job.getConfiguration().set("mapred.mapper.right.table","/genre.txt");
boolean result=job.waitForCompletion(true);
```

基本类型

- Interface Writable

```
public void readFields(DataInput in) throws IOException
 {
        id=in.readInt();
        gender=in.readChar();
        age=in.readInt();
        occid=in.readInt();
        zipcode=Text.readString(in);
 }
public void write(DataOutput out) throws IOException{
        out.writeInt(id);
        out.writeChar(gender);
        out.writeInt(age);
        out.writeInt(occid);
        Text.writeString(out,zipcode);
}
```

基本类型

- **Interface WritableComparable**
  - BytesWritable, LongWritable,NullWritable, VLongWritable

```
public int compareTo(TextPair obj){
        return 0;
}
public void readFields(DataInput in) throws IOException
 {
        key=Text.readString(in);
        table=in.readInt();
 }
public void write(DataOutput out) throws IOException{
        Text.writeString(out, key);
        out.writeInt(table);
}
```

全局数据

- Configuration
  - 适合小数据量(配置)的分发
- Distributed Cache
  - 分发配置文件
  - 为重复分发优化(Cache)
- 没有全局变量
  - 但可以借助zookeeper等系统实现

File Format

- KeyValueTextInputFormat
  - 分隔符(tab)分为两部分，第一部分为key，剩下的部分为value；如果没有分隔符，整行作为 key，value为空
- Text InputFormat/OutputFormat
  - 以行纪录偏移作为key，行数据作为value
  - 数据压缩时整个文件交给一个Map处理
- SequenceFile InputFormat/OutputFormat
  - 以<key,value>序列化方式获取数据。
  - 数据压缩性能较好

练习1

- 实现User表**WritableComparable**对象
- 实现Genre表**WritableComparable**对象
- 实现Movie 表**WritableComparable**对象
- SequenceFile InputFormat/OutputFormat
  - 获取DB数据写入HDFS
  - 从HDFS读取数据打印

InputSplit

- Hadoop Map/Reduce为每个InputSplit 产生一个对应的Map 任务
- 数据大小
- 在文件中的起始位置
- 在文件中的偏移位置
- 存储的datanode hostname
- 根据文件指针偏移和hostname定位Split数据块

RecordReader

- 顺序读取数据单元
- LineRecordReader
  - 根据InputSplit提供的文件指针定位信息，按照行读取每条记录。
  - 以数据行的偏移量作为Key，行数据作为Value输出
  - 将InputSplit提供的字节数据存储样式转化为Map/Reduce可识别的<Key,Value>样式

FileInputFormat

- 根据JobConf初始化InputSplit
- 利用InputSplit创建RecordReader
- 根据RecordReader将字节文件数据转化为可遍历的<Key,Value>数据提供给Hadoop Map/Reduce方法使用
- 检查文件扩展名如果是gz或者lzo则进行解压缩

Combiner

- combiner原理
  - Map output:{<k1,v11>,<k1,v12>,<k1,v13>}
  - Combine:{<k1,[v11,v12,v13]>}
  - Reduce input:{<k1,[v11,v12,v13]>}
- Combiner作用
  - IO优化
  - 尽可能合并
- Combiner使用注意
  - Combiner默认使用reduce对象时问题

Partitioner

- Partitioner 作用
  - Map 数据路由分发到reduce
  - 数据均衡分布计算，避免数据倾斜
- Partitioner 原理

```
/** Partition keys by their {@link Object#hashCode()}. */
public class HashPartitioner<K2, V2> implements Partitioner<K2, V2> {
  public void configure(JobConf job) {}
  /** Use {@link Object#hashCode()} to partition. */
  public int getPartition(K2 key, V2 value,
                          int numReduceTasks) {
    return (key.hashCode() & Integer.MAX_VALUE) % numReduceTasks;
  }
}
```

GroupingComparator

- 原理
- 作用
  - 相同的key 在reduce阶段分组操作

```
public static class CommonJoinComparator extends WritableComparator {

    public CommonJoinComparator() {
       super(TextPair.class, true);
    }
    @SuppressWarnings("unchecked")
    public int compare(WritableComparable a, WritableComparable b) {
       TextPair t1 = (TextPair) a;
       TextPair t2 = (TextPair) b;
       return t1.getKey().compareTo(t2.getKey());
    }
}
```

Map/reduce Report

- Report
  - 用于Map/Reduce应用程序报告进度，设定应用级别的状态消息，更新Counters（计数器）的机制
- 应用
  - **incrCounter**(Enum<?> key, long amount)
  - **incrCounter**(String group, String counter, long amount)
  - **getCounter**(Enum<?> name)
  - **getCounter**(String group, String name)

Map/reduce Counter

- Counter
  - 用于运行监控，调试
- 注意mapred和mapreduce不同的变化
- 通过web和console观察输出
- GrepCount Demo
  - select count(*) from movie where year>1990 and name like '%the%';
  - select count(*) from movie where year<=1990;
  - 观察Newegg={Cond和NoCond参数变化}

Map/Reduce distributed Cache

- DistributedCache
  - 将具体应用相关的、大尺寸的、只读的文件有效地分布放置
  - Map/Reduce框架提供的功能，能够缓存应用程序所需的文件（包括文本，档案文件，jar文件等）
  - Map-Redcue框架在作业执行之前会把必要的文件拷贝到slave节点上。它运行高效是因为每个作业的文件只拷贝一次，并且为那些没有文档的slave节点缓存文档

Map/Reduce distributed Cache

- DistributedCache
  - DistributedCache.addCacheArchive(new URI("/myapp/map.zip", job);
  - DistributedCache.addFileToClassPath(new Path("/myapp/mylib.jar"), job);
  - DistributedCache.addCacheArchive(new URI("/myapp/mytar.tar", job);
  - DistributedCache.addCacheArchive(new URI("/myapp/mytgz.tgz", job);
  - DistributedCache.addCacheArchive(new URI("/myapp/mytargz.tar.gz", job);
  - Path localArchives = DistributedCache.getLocalCacheArchives(job);
  - Path localFiles = DistributedCache.getLocalCacheFiles(job);

Map/Reduce排序

- 如何实现排序？
  - Map 输出排序
  - Reduce处理前排序

Map/reduce 排序算法1

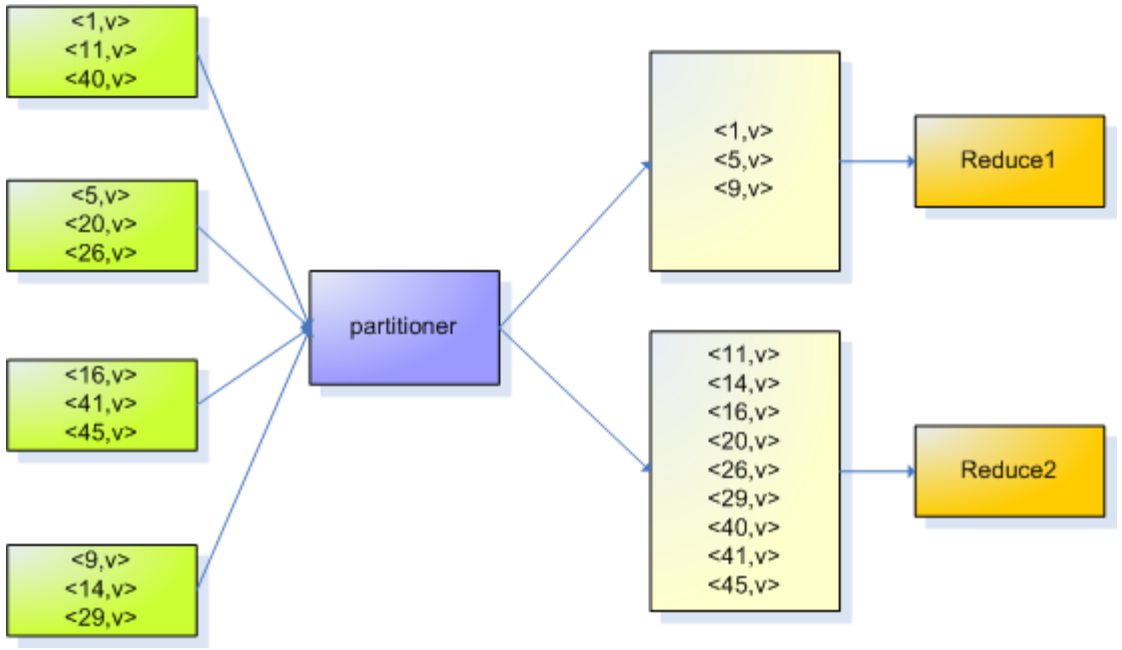
- 利用Map/Reduce Sort实现排序
- 原理:
  - 在reduce处理之前，对key进行排序处理。
  - 在某个reduce任务中，key是有序的。
  - 如果设置reduce数量为1，则可实现排序。
- 缺点:
  - 可能导致性能问题(数据倾斜)

Map/reduce 排序算法2

- 多个Reduce排序
- 利用Partitioner实现
- 原理
  - 有序分配key 复制到各自reduce
  - 保证map阶段结束后从内存或者本地磁盘上向reduce拷贝的<key,value>数据的key分布是有序分配的。

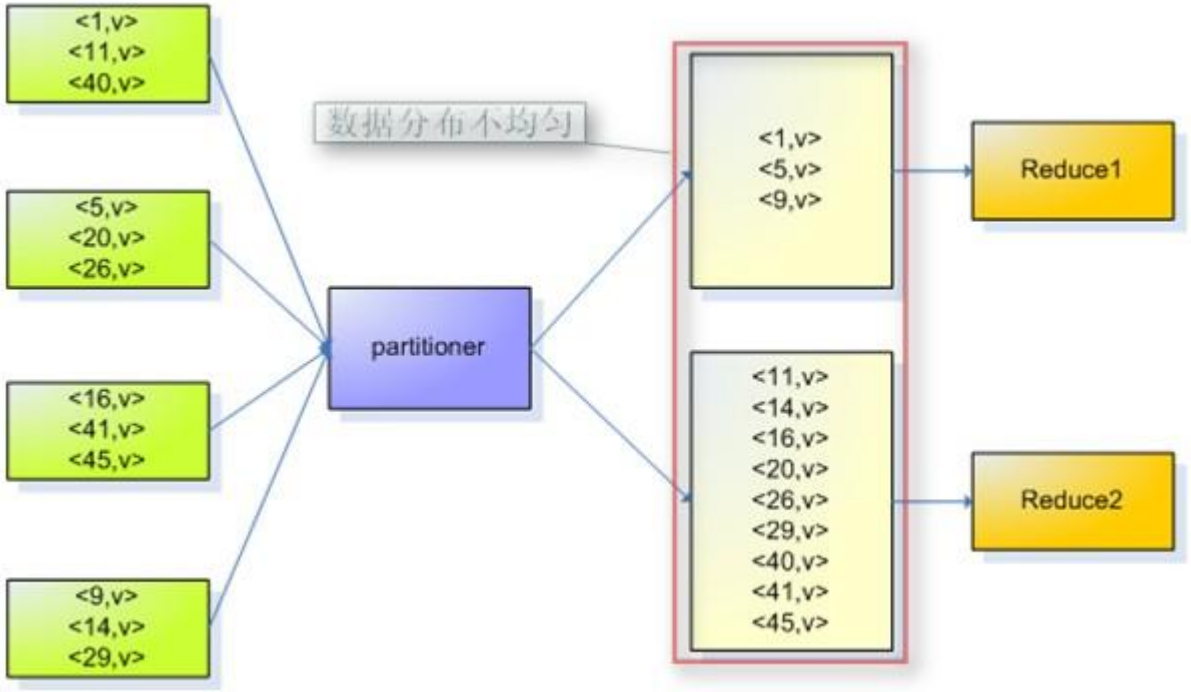
Partitioner 分区

利用Partitioner路由功能实现有序分配



Partitioner 分区

- 数据倾斜是如何产生的？



数据采样

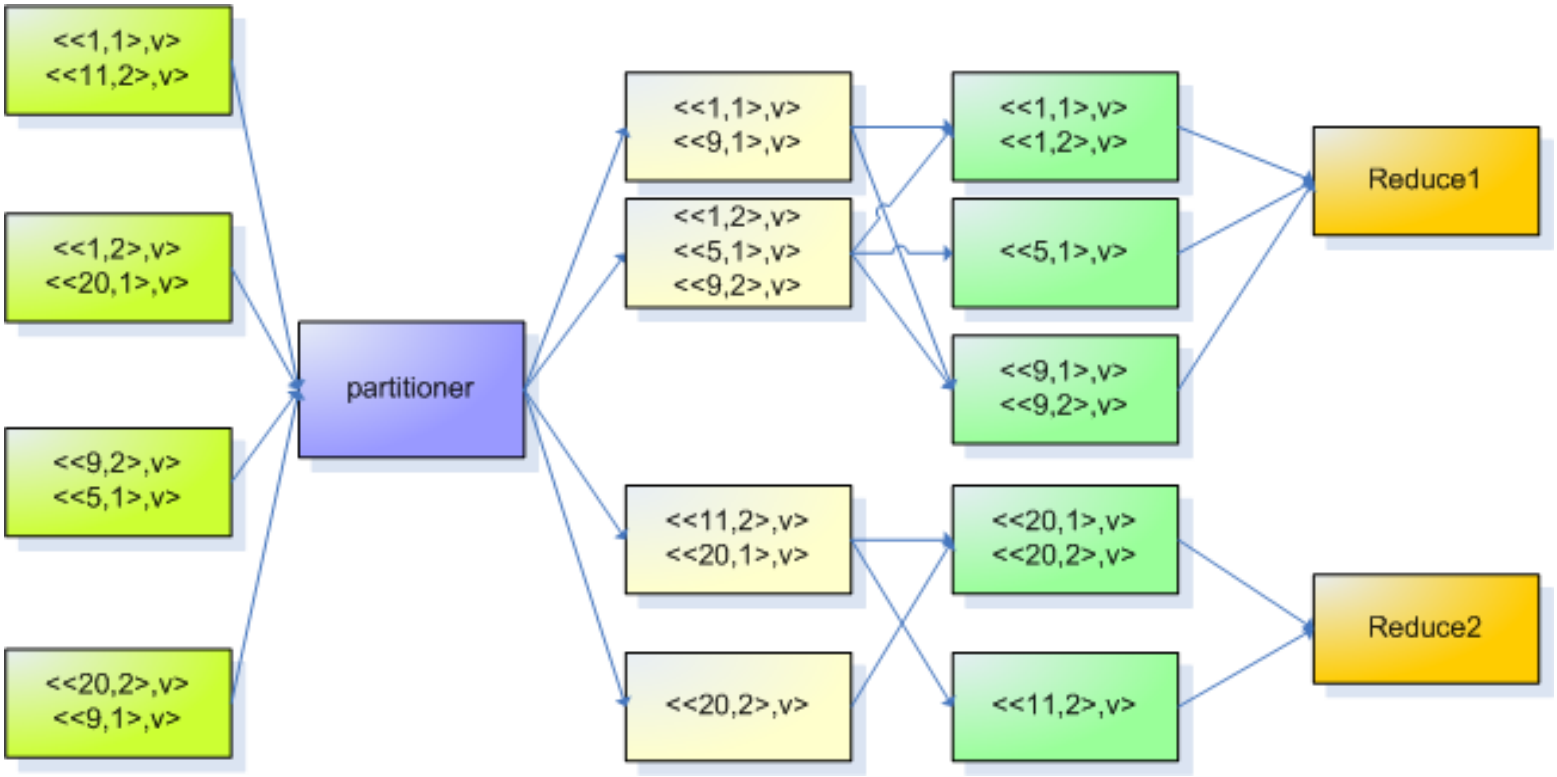
- 数据采样
- 作用
  - 避免对某个Reduce过多的数据和计算任务分配
  - 解决数据倾斜问题
- 数据采样算法
  - SplitSampler
  - RandomSampler
  - IntervalSampler
- http://thinkinginhadoop.iteye.com/blog/709986

Map/reduce 二次排序

- setGroupingComparatorClass
  - 相同key的记录排序
  - 定义聚合比较器（grouping comparator）
  - 比较那些记录被分配到同一个reduce方法
- 用途
  - Join算法应用

分区和分组

- Partitioner和GroupingComparator



Map/reduce Join算法

- 如何在Map/Reduce内实现Join算法？
  - 利用partitioner归并相同的key到同一个reduce任务
  - 利用GroupingComparator归并比较相同key但来自不同table的排序。
  - 在reduce中遍历<<key,table>,value>,合并相同key不同表的数据。

Map/reduce Grep算法

- Map
  - Grep匹配，获取匹配条件的记录
  - 以分组的字段作为key输出，value=1
  - Combine尽量合并
- Reduce
  - 合并相同key所对应的value。
  - 根据分组字段输出Grep匹配数量

Hadoop Streaming

- 调试
- Demo
- 支持多种语言，开发效率高
- 性能

Map/reduce优化

- 优化Map/Reduce作业逻辑
- IO优化
  - combine优化
  - HashMap Combine 优化
    - 针对<k,v1>,<k,v2> => {k,[v1,v2]}
  - 压缩优化

Map/Reduce优化

- 配置优化
  - 根据运行历史进行参数配置调优
    - **Vaidya**
  - Map数量
    - Key输入范围和数据分布
  - Reduce数量
  - 增加JVM内存
    - 尽量让map阶段计算和数据输出都在内存处理
    - Map阶段输出排序消耗内存

Map/Reduce优化

- Map/reduce调试
  - Log
  - Counter
- Hadoop Streaming 调试
  - 标准输入输出调试
- InputFormat/RecordReader/…
  - 本地调试运行