火龙果•整理
uml.org.cn

Hive和Hadoop作业调优

核心系统研发-海量数据　无影

火龙果•整理
uml.org.cn

自我介绍

- 2010年4月 加入阿里云-云梯团队
- 2011年11月 随团队转入集团技术部
- 从事Hadoop的开发、测试、异常调查、云梯运维支持
- 对MapReduce框架及调优有深入研究

追風堂

云梯/Hadoop系列课程

	课程名	讲师
Hive系列	Hive编程入门课程	天照
	Hive高级编程	天照
	Hive实现原理	周忱
Hadoop入门	Hadoop及实践	天照
	云梯1 Hadoop集群架构和服务简介	无谓
Hadoop实践	HDFS介绍及云梯1使用实践	彦唐
	MapReduce编程介绍及云梯1实践	无影
	Hive和MapReduce作业调优	无影
Hadoop原理	Hadoop HDFS实现原理	彦唐
	Hadoop MapReduce实现原理	无影
	Hadoop NameNode单点瓶颈难题和ADFS介绍	济万
	Hadoop JobTracker HA的实现	无影

火龙果•整理
uml.org.cn

课程目标

- **面向学员：**
  - 有编写MapReduce作业经历
  - 希望熟悉作业优化方案
- **课程目标：**
  - 熟悉MapReduce作业常用调优方法
  - 熟悉如何自定义模块

大纲

1. 配置参数调优
2. 自定义模块
3. 编程技巧
4. Profiling

追風堂

火龙果•整理
uml.org.cn

配置参数调优

火龙果•整理
uml.org.cn

配置参数调优

1. 控制Map个数
2. 合理设置Reduce个数
3. 压缩中间数据
4. 减少磁盘IO

追風堂

火龙果•整理
uml.org.cn

配置参数调优 控制Map个数

- Map的个数取决于对输入切割(split)的份数
- 压缩文件不可切割
  - Map数等于输入的文件数
- 非压缩文件及Sequence文件可切割
- 云梯上（HDFS）的一个文件物理上以一个或多个block存储
- 一个block对应一个Linux文件
- block大小由dfs.block.size配置决定 默认：64MB

配置参数调优 控制Map个数

- 查看云梯上一个文件block大小方法：

$ bin/hadoop fs -stat %o */history/auditlog/2012-04-17/10_46_30*

268435456

1. 文件统计信息
2. $ bin/hadoop fs –help stat

输出64MB

```
<property>
  <name>dfs.http.address</name>
  <value>hdpnn:50070</value>
</property>
```

- 查看云梯上一个文件的block个数

$ **bin/hadoop fsck** */history/auditlog/2012-04-17/10_46_30* **-files -blocks**

/history/auditlog/2012-04-17/10_46_30 524288197 bytes, 2 block(s): OK

0. blk_6818414623683754707_6818687352 len=268435456 repl=3

1. blk_1703312378178171970_6818689001 len=255852741 repl=3

火龙果•整理
uml.org.cn

配置参数调优 控制Map个数

- old-mr-api 的切割大小的影响参数
  - mapred.map.tasks 期望的map个数 默认值：1
  - mapred.min.split.size 切割出的split最小size 默认：1
- old-mr-api的切割大小算法
  - splitSize = **max**[minSize, **min**(goalSize, blockSize)]
  - minSize = ${mapred.min.split.size}
  - goalSize = totalSize **/** ${mapred.map.tasks}
- mapred.map.tasks可以增大map数
- mapred.min.split.size可以减少map数

火龙果•整理
uml.org.cn

配置参数调优 控制Map个数

- new-mr-api 的切割大小的影响参数
  - mapred.max.split.size 切割出的split最大size 默认：Long.MAX_VALUE
  - mapred.min.split.size 切割出的split最小size 默认：1
- old-mr-api的切割大小算法
  - splitSize = **max**[minSize, **min**(maxSize, blockSize)]
  - minSize = ${mapred.min.split.size}
  - maxSize = ${mapred.max.split.size}
- mapred.max.split.size可以增大map数
- mapred.min.split.size可以减少map数

火龙果•整理
uml.org.cn

配置参数调优 控制Map个数

- map执行时间小于30s-40s，应减少map数
- 调度开销1-5秒
- 组内资源不足限制并行度
- FileInputFormat 1个文件至少需要1个map
  - TextInputFormat
  - SequenceFileInputFormat
- 一个Map处理多个文件需要更换InputFormat
  - MultiFileInputFormat (deprecated)
  - CombineFileInputFormat

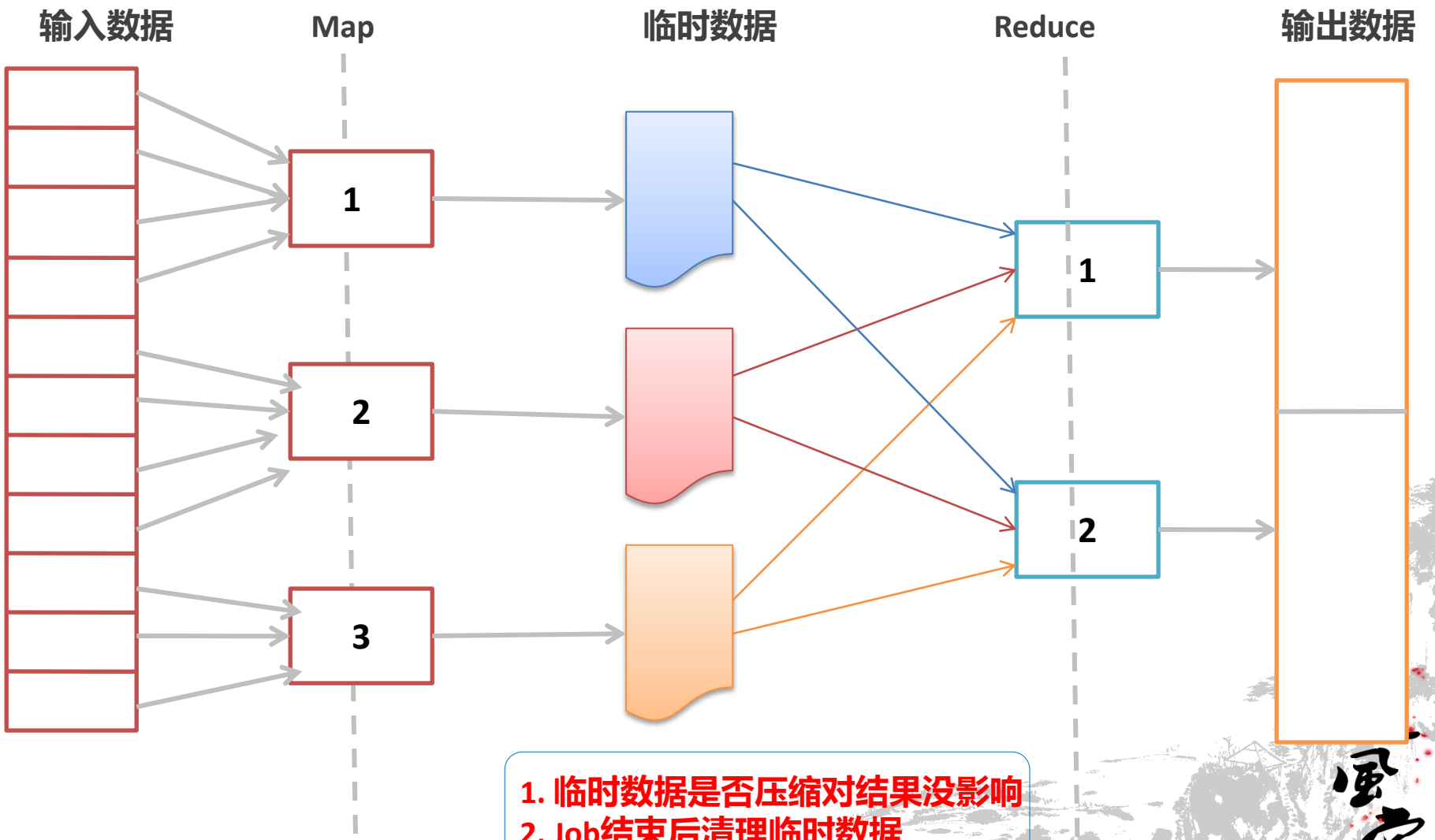
火龙果•整理
uml.org.cn

配置参数调优 设置Reduce个数

- Reduce个数设置
  - mapred.reduce.tasks 默认值：1
- Reduce个数太小
  - 单个reduce task执行很慢
  - 出错重试开销很大
- Reduce个数设置太大
  - Shuffle开销（m*r）及调度开销很大
  - Job输出大量小文件，影响后续Job
- 推荐Reduce个数
  - 单个reduce task处理数据量介于 1~10G之间
  - Reduce个数要少于Map个数

火龙果•整理
uml.org.cn

配置参数调优 压缩中间数据



1. 临时数据是否压缩对结果没影响
2. Job结束后清理临时数据

火龙果•整理
uml.org.cn

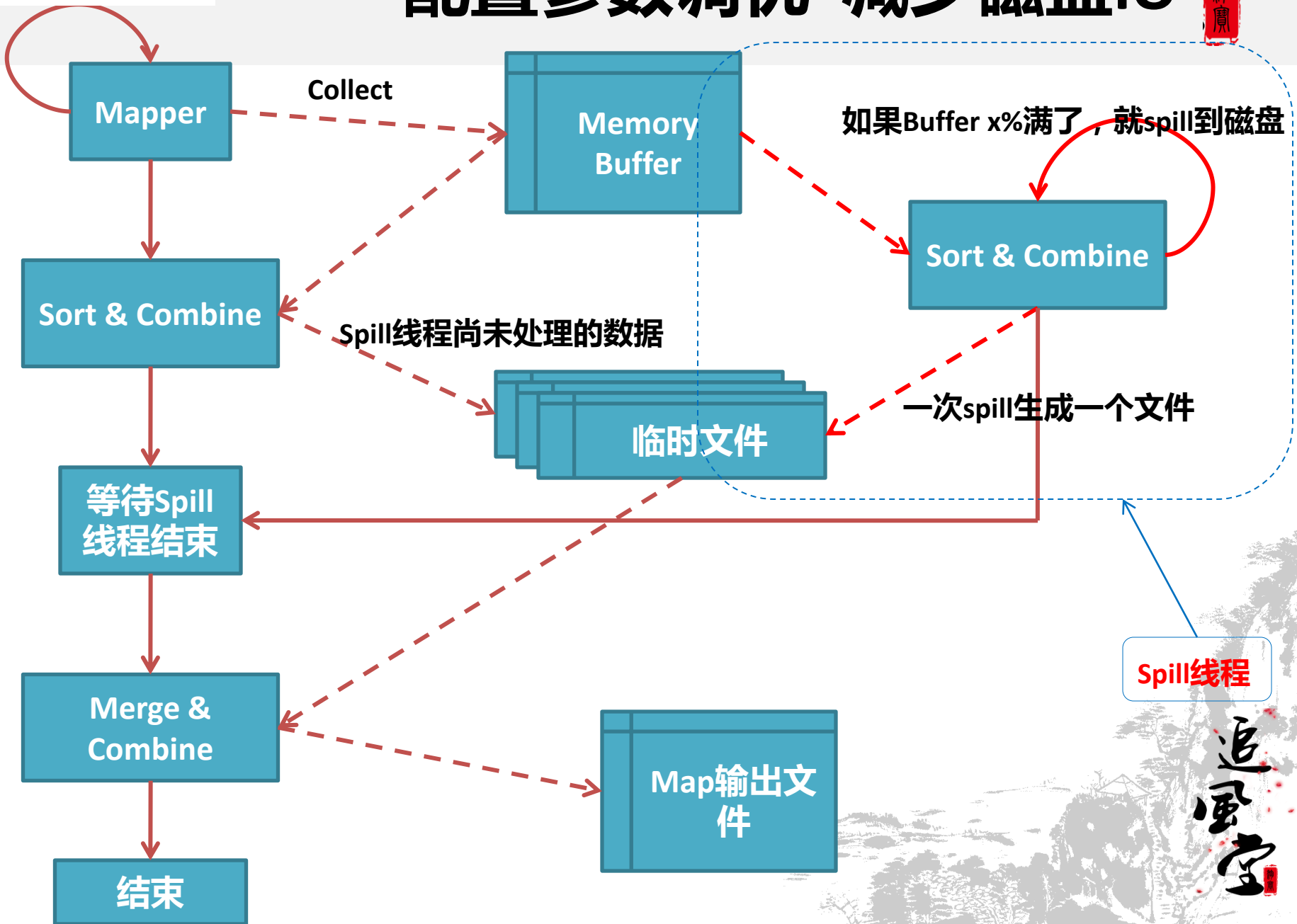
配置参数调优 压缩中间数据

- 压缩中间数据的好处
  - 减少磁盘操作
  - 减少网络传输数据量
- 配置方法：
  - mapred.compress.map.output 设为true
  - mapred.map.output.compression.codec
    - org.apache.hadoop.io.compress.LzoCodec
    - org.apache.hadoop.io.compress.SnappyCodec

淘宝数据平台生产作业用Snappy

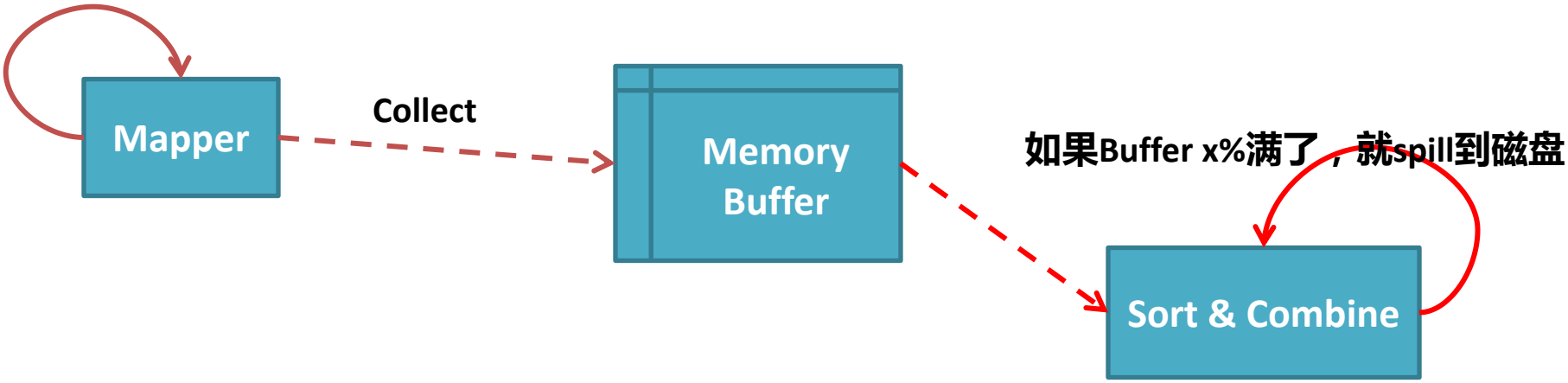
火龙果•整理
uml.org.cn

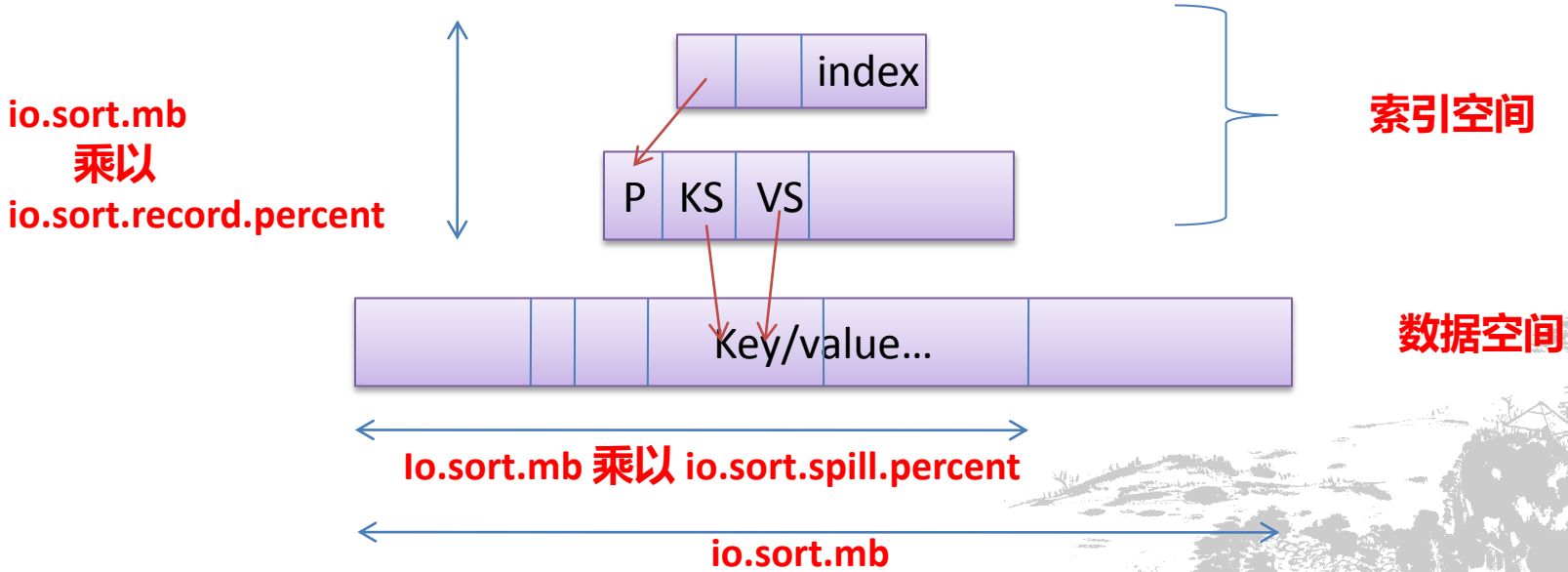
配置参数调优 减少磁盘IO



火龙果•整理
uml.org.cn

配置参数调优 减少磁盘IO





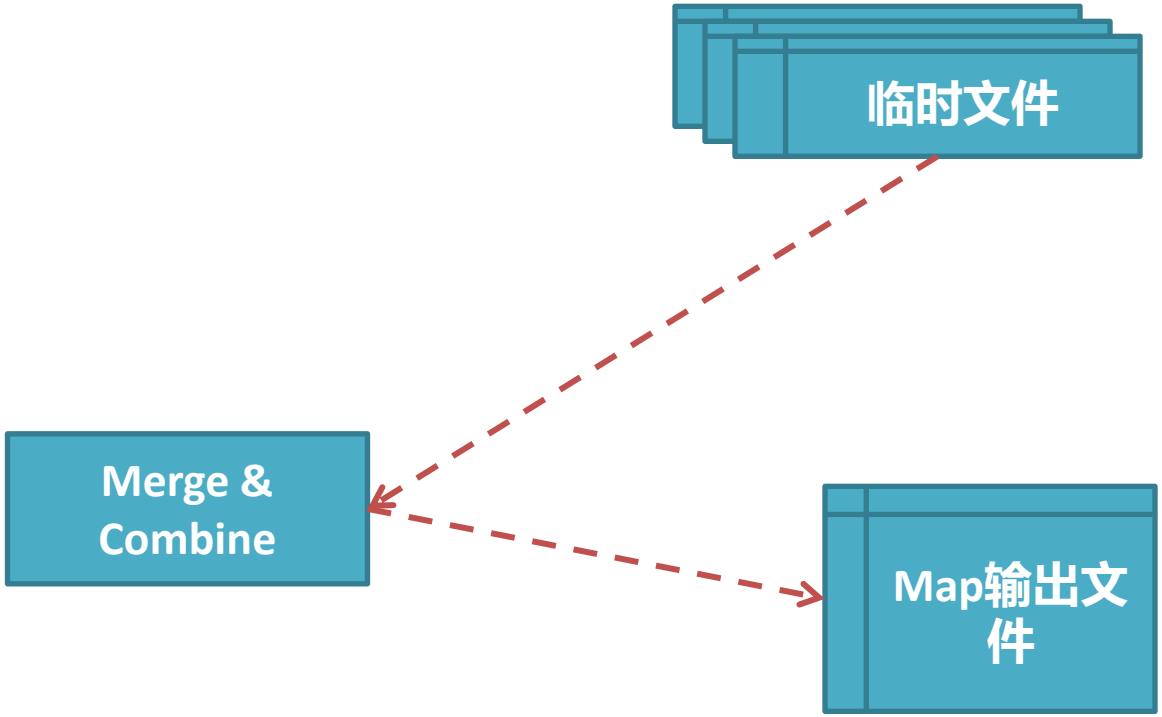
火龙果•整理
uml.org.cn

配置参数调优 减少磁盘IO

- io.sort.mb
  - 排序空间大小 默认值：100m
  - 需要与mapred.child.java.opts相配 默认：-Xmx200m
- io.sort.record.percent
  - 索引空间所在比例，默认：0.05
  - Spilling map output: record full = true 需要调大
  - Spilling map output: buffer full = true 需要调小
- io.sort.spill.percent
  - 触发spill的内存占用比 默认：0.8

火龙果•整理
uml.org.cn

配置参数调优 减少磁盘IO



- io.sort.factor
  - 外排时同时merge的文件数， 默认：10
  - Reduce端同样适用
  - 建议调大为100

火龙果•整理
uml.org.cn

配置参数调优 总结

- 控制Map个数，减轻调度开销
- 合理设置Reduce个数，避免单个task数据量处理过大或过少
- 启用中间数据压缩，用CPU换磁盘和网络
- 尽量减少磁盘IO，磁盘操作是最慢的

追風堂

火龙果•整理
uml.org.cn

自定义模块

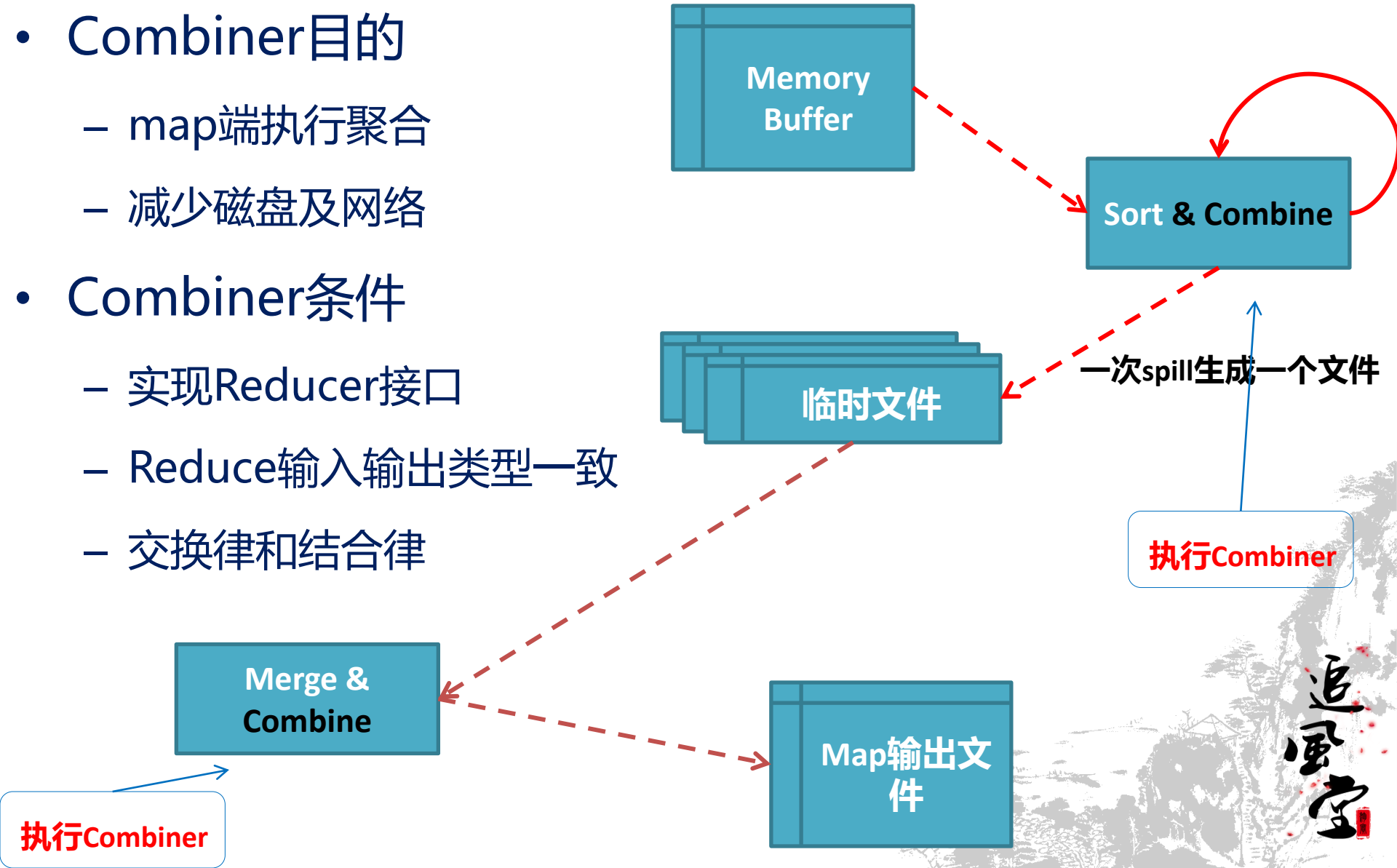
火龙果•整理
uml.org.cn

自定义模块

1. Combiner
2. Partitioner
3. Comparator
4. Multi|CombineFileInputFormat
5. KeyValueTextInputFormat
6. Secondary Sort

自定义模块 Combiner

- Combiner目的
  - map端执行聚合
  - 减少磁盘及网络
- Combiner条件
  - 实现Reducer接口
  - Reduce输入输出类型一致
  - 交换律和结合律



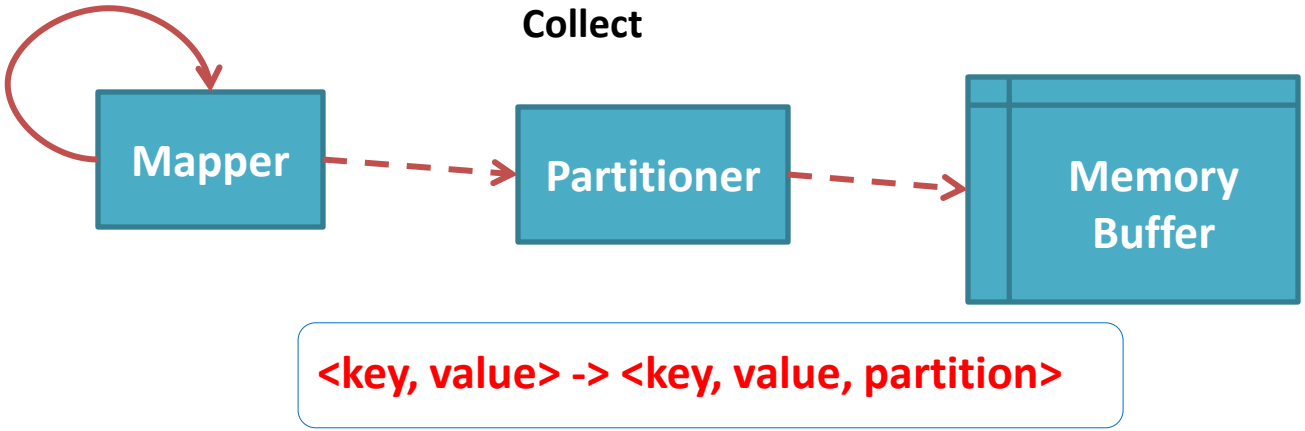
火龙果•整理
uml.org.cn

自定义模块 Combiner

- Combiner使用方法
  - conf.setCombinerClass(Reduce.class);
  - mapred.combiner.class
- Combiner适用场景
  - *Reduce input groups*远小于*Reduce input records*
  - Map输出数据量较大
  - 启用Combiner后较少20%以上map本地输出

火龙果•整理
uml.org.cn

自定义模块 Partitioner



- Partitioner作用
  - 将map输出的<key,value>对分配给相应的reduce
  - 所有的<key,value>对都需要分配给相应的reduce
  - 相同的key分配给同一个reduce
  - 每次分配不依赖于其他<key,value>

自定义模块 Partitioner

- Partitioner接口
  - configure(JobConf)
  - int getPartition(key, value, numPartitions)

[0, numPartitions)

map输出的一个<key,value>对

reduce的个数

- 默认实现 HashPartitioner

```
/** Use {@link Object#hashCode()} to partition. */
public int getPartition(K2 key, V2 value,
                        int numReduceTasks) {
  return (key.hashCode() & Integer.MAX_VALUE) % numReduceTasks;
}
```

输入数据均匀分配，则输出也会均匀分布

自定义模块 Partitioner

```
public static class MyPartitioner implements Partitioner<Text, IntWritable> {

    @Override
    public void configure(JobConf job) {}

    @Override
    public int getPartition(Text key, IntWritable value, int numReduceTasks) {
        if (key.toString().equals("taobao")) {
            return 0;
        }
        return (key.hashCode() & Integer.MAX_VALUE )
                % (numReduceTasks - 1) + 1;
    }
}
```

含有"taobao"的key由0号reduce处理

其他key由其他reduce处理

- 启用自定义Partitioner
  - conf.setPartitionerClass(MyPartitioner.class)

火龙果•整理
uml.org.cn

自定义模块 Comparator

- RawComparator接口
  - int compare(byte[] b1, int s1, int l1, byte[] b2, int s2, int l2)
  - 用以实现对Key排序
- Hadoop自带类型均有定义好的Comparator
- Hadoop自带的Comparator针对特定类型做了优化
- Text的Comparator
  - 比较字符串的长度
  - 逐个字节比较序列化后的数据

自定义模块 Comparator

```
public static class MyComparator implements RawComparator<Text> {

    @Override
    public int compare(byte[] b1, int s1, int l1, byte[] b2, int s2, int l2) {
        return b1[s1] - b2[s2];
    }

    @Override
    public int compare(Text o1, Text o2) {
        return 0;
    }
}
```

1. 只比较第一个字节
2. 第一个字节相同的key被认为是同一个key

- 启用自定义Comparator
  - conf.setOutputKeyComparatorClass(MyComparator.class);

火龙果•整理
uml.org.cn

自定义模块 MultiFileInputFormat

- MultiFileInputFormat
  - 重写getSplits()方法，即切割输入的算法
  - 由mapred.map.tasks确定split的个数
  - 一个文件只由一个map处理
  - 一个map可以处理多个文件
- 启用MultiFileInputFormat
  - 自己实现getRecordReader()方法
  - 参考：MultiFileWordCount

火龙果•整理
uml.org.cn

自定义模块 CombineFileInputFormat

- CombineFileInputFormat
  - 重写getSplits()方法，即切割输入的算法
  - MultiFileInputFormat的升级版
  - **mapred.max.split.size** 默认：0
  - mapred.min.split.size.per.node 默认：0
  - mapred.min.split.size.per.rack 默认：0
- 启用CombineFileInputFormat
  - 自己实现getRecordReader()方法
  - 参考：社区0.21版本的MultiFileWordCount

0：同一Rack上的block构成一个split

>0：同一Node上的block构成一个split；剩下的block和同一rack的block构成一个split

火龙果•整理
uml.org.cn

自定义模块 KeyValueTextInputFormat

- KeyValueTextInputFormat
  - 重写RecordReader
  - 将文件中的一行视为<key, value>对
  - Key.value.separator.input.file key&value分隔符，默认：‘\t’
  - 如果没有找到分隔符，则key为一行，value为空字符串
  - Map的输入类型为<Text, Text>
- 启用KeyValueTextInputFormat
  - conf.setInputFormat(KeyValueTextInputFormat.class);

Map的输入类型取决于InputFormat

自定义模块 Secondary Sort

- Reduce输入数据特征
  - 每个Reduce输入的Key是有序的
  - 每个Reduce输入的Value是无序的
  - 每个Reduce输入的Value顺序不固定
- Secondary Sort目标
  - 每个Reduce输入的Value是有序的

1. 同一个Job多次运行
2. 与map完成顺序有关

自定义模块 Secondary Sort

- 场景例子
  - 每个公司最大部门人数

公司名	某部门人数	公司名	某部门人数
Taobao	52	B2B	120
Taobao	31	B2B	72
Taobao	67	Aliyun	13
Alipay	10	Aliyun	32
Alipay	36	Aliyun	3
Alipay	29		

自定义模块 Secondary Sort

- 自定义key类型
  - 将value放入key中
  - 用NullWritable作为value类型

```
9  public class MyKey implements WritableComparable<MyKey> {
10   public final Text first;
11   public final IntWritable second;
12
13   public MyKey() {
14     first = new Text();
15     second = new IntWritable();
16   }
17
18   public MyKey(Text first, IntWritable second) {
19     this.first = first;
20     this.second = second;
21   }
```

1. first存放公司名
2. second存放某部门人数

自定义模块 Secondary Sort

- 自定义key类型
  - 实现Writable接口，支持序列化

```
23  @Override
24  public void write(DataOutput out) throws IOException {
25    first.write(out);
26    second.write(out);
27  }
28
29  @Override
30  public void readFields(DataInput in) throws IOException {
31    first.readFields(in);
32    second.readFields(in);
33  }
```

自定义模块 Secondary Sort

- 自定义key类型
  - 实现Comparable，支持比较
  - 先排first（升序），后排second（降序）

```
40  @Override
41  public int compareTo(MyKey tp) {
42    int cmp = first.compareTo(tp.first);
43    if (cmp != 0) {
44      return cmp;
45    }
46    return -second.compareTo(tp.second);
47  }
```

降序

自定义模块 Secondary Sort

- 自定义key类型
  - 实现toString()方法，用以输出

```
35  @Override
36  public String toString() {
37    return first + "\t" + second;
38  }
```

自定义模块 Secondary Sort

- 自定义Partitioner
  - first相同的key分配给同一个reduce

```
6  public class MyPartitioner
7    implements Partitioner<MyKey, NullWritable> {
8
9    @Override
10   public void configure(JobConf job) {}
11
12   @Override
13   public int getPartition(MyKey key, NullWritable value,
14                           int numPartitions) {
15     return (key.first.hashCode() & Integer.MAX_VALUE) % numPartitions;
16   }
17 }
```

根据公司名做分区

自定义模块 Secondary Sort

- 自定义ValueGroupingComparator
  - first相同的key归并到一起
  - 默认使用Key的Comparator

```
5  public class MyGroupComparator extends WritableComparator {
6    protected MyGroupComparator() {
7      super(MyKey.class, true);
8    }
9
10   @Override
11   public int compare(WritableComparable w1, WritableComparable w2) {
12     MyKey m1 = (MyKey) w1;
13     MyKey m2 = (MyKey) w2;
14     return m1.first.compareTo(m2.first);
15   }
16 }
```

根据公司名归并

自定义模块 Secondary Sort

- 输入数据 格式 公司名+空格+某部门人数

$ bin/hadoop fs -cat */group/dc/liangly/list*

Taobao 52

Taobao 31

Taobao 67

Alipay 10

Alipay 36

Alipay 29

B2B 120

B2B 72

Aliyun 13

Aliyun 32

Aliyun 3

自定义模块 Secondary Sort

- Map实现

```
15  public static class MyMap extends MapReduceBase
16    implements Mapper<Text, Text, MyKey, NullWritable> {
17
18    private IntWritable num = new IntWritable();
19
20    @Override
21    public void map(Text key, Text value,
22                    OutputCollector<MyKey, NullWritable> output,
23                    Reporter reporter) throws IOException {
24
25      num.set(Integer.parseInt(value.toString()));
26      MyKey myKey = new MyKey(key, num);
27      output.collect(myKey, NullWritable.get());
28    }
29  }
```

key：公司名
value：某部门人数

构造MyKey对象

value不用

自定义模块 Secondary Sort

- Reduce实现

```
31  public static class MyReduce extends MapReduceBase
32    implements Reducer<MyKey, NullWritable, MyKey, NullWritable> {
33
34    @Override
35    public void reduce(MyKey key, Iterator<NullWritable> values,
36                       OutputCollector<MyKey, NullWritable> output,
37                       Reporter reporter) throws IOException {
38      output.collect(key, NullWritable.get());
39    }
40  }
```

key中包含了公司名及最大部门人数

value不用

自定义模块 Secondary Sort

- 配置自定义的模块

配置输出的Key&Value类型

配置Partitioner

配置归并比较类

```
48    conf.setOutputKeyClass(MyKey.class);
49    conf.setOutputValueClass(NullWritable.class);
50
51    conf.setMapperClass(MyMap.class);
52    conf.setReducerClass(MyReduce.class);
53
54    conf.setPartitionerClass(MyPartitioner.class);
55
56    conf.setOutputValueGroupingComparator(MyGroupComparator.class);
57
58    conf.setInputFormat(KeyValueTextInputFormat.class);
59    conf.set("key.value.separator.in.input.line", " ");
60
61
62    FileInputFormat.setInputPaths(conf, new Path(args[0]));
63    FileOutputFormat.setOutputPath(conf, new Path(args[1]));
```

配置InputFormat

自定义模块 Secondary Sort

```
$ bin/hadoop jar job.jar MySecondarySort \
> -Dmapred.map.tasks=3 \
> -Dmapred.reduce.tasks=2 \
> /group/dc/liangly/list \
> /group/dc/liangly/out
```

```
$ bin/hadoop fs -cat /group/dc/liangly/out/*
Alipay  36
Aliyun  32
B2B     120
Taobao  67
```

火龙果•整理
uml.org.cn

自定义模块 Secondary Sort

- Secondary Sort总结
  - 自定义一个KEY类型封装原始的key和value
  - KEY的比较器，先比较原始key后比较原始value
  - 自定义Partitioner，仅根据原始key分区
  - 自定义归并比较器，仅根据原始key归并

由于reduce不满足交换率，不能作为Combiner！

火龙果•整理
uml.org.cn

自定义模块 总结

- 启用Combiner在map端做归并
- 自定义Partitioner控制<key,value>的分区
- 自定义Comparator控制比较方法
- 自定义InputFormat可以控制输入切割方法
- 通过自定义模块实现Secondary Sort

追風堂

火龙果•整理
uml.org.cn

编程技巧

编程技巧

1. 重用Writable对象
2. 慎用String.split()
3. 使用StringBuilder
4. 谨慎Catch异常

追风堂

编程技巧 重用 Writable对象

```
16  public static class TokenizerMapper extends MapReduceBase
17    implements Mapper<LongWritable, Text, Text, IntWritable> {
18
19    private final static IntWritable one = new IntWritable(1);
20    private Text word = new Text();
21
22    public void map(LongWritable key, Text value,
23                    OutputCollector<Text, IntWritable> output,
24                    Reporter reporter) throws IOException {
25      String line = value.toString();
26
27      StringTokenizer itr = new StringTokenizer(line);
28      while (itr.hasMoreTokens()) {
29        word.set(itr.nextToken());
30        output.collect(word, one);
31      }
32    }
33  }
```

定义Mapper的成员变量

只改变word内容，而不用new对象

编程技巧 慎用String.split()

```
23    public void map(LongWritable key, Text value,
24                    OutputCollector<Text, IntWritable> output,
25                    Reporter reporter) throws IOException {
26      String line = value.toString();
27      String[] parts = line.split(" ", 0);
28
29      for (String part : parts) {
30        word.set(part);
31        output.collect(word, one);
32      }
33    }
34  }
```

频繁执行String.split()可 优化

```
public String[] split(String regex, int limit) {
    return Pattern.compile(regex).split(this, limit);
}
```

JDK中String.split()方法的实现

编程技巧 慎用String.split()

```
23    private Pattern pattern = Pattern.compile(" ");
24
25    public void map(LongWritable key, Text value,
26                    OutputCollector<Text, IntWritable> output,
27                    Reporter reporter) throws IOException {
28      String line = value.toString();
29      String[] parts = pattern.split(line, 0);
30
31      for (String part : parts) {
32        word.set(part);
33        output.collect(word, one);
34      }
35    }
```

定义Mapper的成员变量

重用pattern对象

- 注意事项：
  - 如果只关注前N个匹配，则将0换成N

编程技巧 使用StringBuilder

```
25    public void map(LongWritable key, Text value,
26                    OutputCollector<Text, IntWritable> output,
27                    Reporter reporter) throws IOException {
28      String line = value.toString();
29      String[] parts = pattern.split(line, 0);
30      String myValue = parts[0] + "\t" + parts[1] + "\t" + parts[2];
31
32      word.set(myValue);
33      output.collect(word, one);
34    }
```

直接拼接会创建大量的字符串对象

1. StringBuilder可减少字符串创建
2. 初始化StringBuilder大小，避免内存缩放

```
25    public void map(LongWritable key, Text value,
26                    OutputCollector<Text, IntWritable> output,
27                    Reporter reporter) throws IOException {
28      String line = value.toString();
29      String[] parts = pattern.split(line, 0);
30      int length = parts[0].length() + parts[1].length() + parts[2].length();
31      StringBuilder sb = new StringBuilder(length + 2);
32      sb.append(parts[0]).append("\t").append(parts[1]);
33      sb.append("\t").append(parts[2]);
34      String myValue = sb.toString();
35
36      word.set(myValue);
37      output.collect(word, one);
38    }
```

编程技巧 谨慎Catch异常

1. 最好不要捕获异常
2. 如果捕获到异常，要再次抛出
3. 捕获到异常不抛出案例
    1. configure()中捕获异常：输出数据比预期少一半
    2. reduce()中捕获异常：Quota超出后作业仍然成功

追風堂

编程技巧 总结

- 重用Writable对象可以大幅减少创建对象次数
- 用Pattern取代String.split()
- 用StringBuilder实现字符串拼接
- 尽量不要捕获异常

追風堂

火龙果•整理
uml.org.cn

PROFILING

火龙果•整理
uml.org.cn

Profiling

- 启用Profiling　　默认：false
  - JobConf.setProfileEnabled(boolean)
  - mapred.task.profile
- Profiling区间　　默认：0-2
  - JobConf.setProfileTaskRange(boolean, String)
  - mapred.task.profile.{maps|reduces}
- 配置Profile参数
  - 默认：hprof=cpu=samples,heap=sites,force=n,thread=y,verbose=n,file=%s
  - JobConf.setProfileParams(String)
  - mapred.task.profile.params

Profiling

- **$ bin/hadoop jar** *hadoop-*-examples.jar wordcount *

*-Dmapred.output.compress=true *

*-Dmapred.task.profile=true *

*-Dmapred.task.profile.maps=500-500 -Dmapred.task.profile.reduces=0-0 *

*-Dmapred.task.profile.params= *

*"-agentlib:hprof=cpu=samples,interval=1000,force=n,thread=y,verbose=n,file=%s" *

*-Dmapred.reduce.tasks=10 *

/history/auditlog/2012-04-07/ /group/dc/liangly/out

1.启用输出数据压缩
2. 默认压缩算法为Zlib

启用Profiling

只对reduce-0 Profiling

1.只采样CPU
2. 采样间隔设为100ms

- 注意事项
  - 启用Profiling的task会变的很慢
  - Profiling结果可以在task日志网页及Gateway上看到

Profiling

```
TRACE 300545: (thread=200001)
        org.apache.hadoop.io.compress.zlib.ZlibCompressor.deflateBytesDirect(ZlibCompressor.java:Unknown line)
        org.apache.hadoop.io.compress.zlib.ZlibCompressor.compress(ZlibCompressor.java:315)
        org.apache.hadoop.io.compress.CompressorStream.compress(CompressorStream.java:76)
        org.apache.hadoop.io.compress.CompressorStream.write(CompressorStream.java:71)
```

main线程的一次TRACE

```
CPU SAMPLES BEGIN (total = 211) Sun Apr 15 16:05:37 2012
rank   self  accum   count trace method
   1 29.86% 29.86%      63 300546 sun.nio.ch.EPollArrayWrapper.epollWait
   2 28.44% 58.29%      60 300545 org.apache.hadoop.io.compress.zlib.ZlibCompressor.deflateBytesDirect
   3  6.16% 64.45%      13 300464 java.net.SocketInputStream.socketRead0
```

汇总的CPU使用热点

HPROF资料：
http://java.sun.com/developer/technicalArticles/Programming/HPROF.html

火龙果•整理
uml.org.cn

总结

- 通过合理配置参数较少IO操作及开销
- 通过自定义模块灵活控制数据处理
- 利用编程技巧可以较少不必要的操作
- 通过Profiling可以分析作业瓶颈

追風堂

火龙果•整理
uml.org.cn

推荐学习资料

- **Hadoop权威指南**

  http://wiki.aliyun-inc.com/projects/apsara/wiki/Yunti1Resources

- **Hadoop MapReduce编程指南**

  http://hadoop.apache.org/common/docs/r0.19.2/mapred_tutorial.html

- **Yahoo! Hadoop博客**

  http://developer.yahoo.com/blogs/hadoop/

- **Cloudera Hadoop博客**

  http://www.cloudera.com/blog/

追風堂

Q&A

追風堂

火龙果•整理
uml.org.cn

谢 谢！

追風堂