（ps:前几日忙于数据库作业忘记更新到 github 上了，只做了简单的笔记，今天一并提交）

2020/6/22 初步了解

Hadoop 是什么

1. Hadoop 就是存储海量数据和分析海量数据的工具。
2. Hadoop 是由 java 语言编写的，在分布式服务器集群上存储海量数据并运行 分布式分析应用 的开源框架，其核心部件是 HDFS 与 MapReduce。
3. HDFS 是一个分布式文件系统：引入存放文件元数据信息的服务器 Namenode 和实际存放数 据的服务器 Datanode，对数据进行分布式储存和读取。
4. MapReduce 是一个计算框架：MapReduce 的核心思想是把计算任务分配给集 群内的服务器里执行。通过对计算任务的拆分（Map 计算/Reduce 计算）再 根据任务调度器 （JobTracker）对任务进行分布式计算。
5. HDFS 为海量的数据提供了存储，则 MapReduce 为海量的数据提供了计算。 把 HDFS 理解为 一个分布式的，有冗余备份的，可以动态扩展的用来存储大 规模数据的大硬盘。把 MapReduce 理解成为一个计算引擎，按照 MapReduce 的规则编写 Map 计算/Reduce 计算的 程序，可以完成计算任务。

使用 Hadoop

1、搭建 Hadoop 集群 无论是在 windows 上装几台虚拟机玩 Hadoop，还是真实 的服务器来玩，说简单点就是把 Hadoop 的安装包放在每一台服务器上，改改配 置，启动就完成了 Hadoop 集群的搭建。
2、上传文件到 Hadoop 集群 Hadoop 集群搭建好以后，可以通过 web页面查看 集群的情况，还可以通过 Hadoop 命令来 上传文件到 hdfs 集群，通过 Hadoop 命令在 hdfs 集群上建立目录，通过 Hadoop 命令删除集 群上的文件等等。
3、编写 map/reduce 程序 通过集成开发工具（例如 eclipse）导入Hadoop 相关 的 jar 包，编写 map/reduce 程序，将程 序打成 jar 包扔在集群上执行，运行后出计算结果。

2020/6/23

了解 MapReduce 以及 Hadoop 分布式文件系统（HDFS）

已掌握知识:

1. map 与 reduce（map 处理 NCDC 数据，提取需要的信息输出，然后通过 mapreduce 框架处理，然后再发送到 reduce 函数，reduce 选择需要的输出）
2. map 函数的实现（通过 Mapper 接口【泛型接口，指定输入键，输入值，输出键，输出值】）
3. LongWriteable 相当于 Long,Text 相当于 String IntWritable 相当于 Integer
4. Reduce 通过 Reducer 接口实现
5. JobConf 对象指定了作业的各种参数。他授予你对整个作业的控制权
6. 新的 API 不兼容以前的 API(以前偏向于接口，现在偏向于抽象类；放的包不一样；新的广泛使用 context；新的 API 同时支持推拉的迭代；新的统一了配置，作业的控制的执行由 Job 类来负责)
7. Mapreduce 的分布化
8. 数据流（用 combiner 减少 map 与 reduce 之间的数据传输量）
9. Hadoop 流（Ruby 语言 -combiner 设置 -file Python 版）
10. Hadoop 管道 C++接口 键和值是字节缓存，表示为 STL 的字符串。
11. 分布式文件系统：管理计算机网络存储 HDFS，以流式数据访问模式存储超大文件（一次写入，多次读取 ；）：低延迟数据访问；大量的小文件；多用户写入，任意修改文件
12. HDFS 的概念
    a) 块：代表一个磁盘能够读写的最小数据量；HDFS 中默认为 64MB【减少寻址开销】（小于一个块的内容不会独占一个）好处方便跨磁盘，利用抽象单元可以简化储存子系统 fsck 显示块的信息
    b) 名称节点与数据节点（名称节点管理者，数据节点工作者）
    c) 防止名称节点故障（复制，而写入 NFS）
    d) 命令行接口 fs-copyFromLocal
    e) 文件系统（org.apache.hadoop.fs.FileSystem）
    f) Thrift API 将 hadoop 文件
    g) 其他接口
13. Java 接口
    a) 读取数据 java.net.URL 打开数据流，从而从中读取数据
    b) SetURLStreamHandler-Factory
    c) 使用 FileSystem API 读取数据【get(Configuraction conf)throws IOException;get(URL uri, Configuration conf)throws IOException】
    d) 写入数据 public FSDataOutputStream create(Path f)throws IOExpection 重载方法 Progressable 告知写入进度或者用 append（）在一个已有文件中追加（public FSDataOutputStream append (Path f)throws IOExpection）

e) 目录 public Boolean mkdirs(Path f)throws IOException

f) 查询文件系统 FileSyetem getFileStatus（）方法，如果不存在会抛出异常

```
public FileStatus[] listStatus(Path f) throws IOException
public FileStatus[] listStatus(Path f, PathFilter filter)
throws IOException
public FileStatus[] listStatus(Path[] files) throws IOException
public FileStatus[] listStatus(Path[] files, PathFilter filter)
throws IOException
```

g) 文件格式

```
public FileStatus[] globStatus(Path pathPattern) throws IOException
public FileStatus[] globStatus(Path pathPattern, PathFilter
filter) throws IOException
```

h) 可选 PathFilter

i)

```
package org.apache.hadoop.fs;

public interface PathFilter {
    boolean accept(Path path);
}
```

j) 删除数据

```
public boolean delete(Path f, boolean recursive) throws IOException
```

14. 数据流
15. 通过 distcp 进行并行复制

distcp 一般用于在两个 HDFS 集群中传输数据。如果集群在 Hadoop 的同一版本上运行，就适合使用 hdfs 方案：

```
% hadoop distcp hdfs://namenode1/foo hdfs://namenode2/bar
```

用-overwrite 和 update 会改变路径和目标路径的含义

16. Hadoop 归档文件

```
% hadoop archive -archiveName files.har /my/files /my
```

第一个选项为归档文件名称

```
% hadoop fs -lsr har:///my/files.har   递归
```

```
% hadoop fs -lsr har:///my/files.har/my/files/dir
% hadoop fs -lsr har://hdfs-
localhost:8020/my/files.har/my/files/dir
```

17. 使用其他文件系统

2020/6/24

Hadoop 的 I/O

常用错误检测代码 CRC-32

- 用 open 传递 false 给 setVerifyChecksum() -get -copyToLocal -ignpreCrc xgexksumFileSystem

压缩

压缩格式	工具	算法	文件扩展名	多文件	可分割性
DEFLATE*	无	DEFLATE	.deflate	不	不
Gzip	gzip	DEFLATE	.gz	不	不
ZIP	zip	DEFLATE	.zip	是	是，在文件范围内
bzip2	bzip2	bzip2	.bz2	不	是
LZO	lzop	LZO	.lzo	不	不

Gzip -1 file(-1,表示速度最优，-9 表示空间最优)

- 编码。解码器

压缩格式	Hadoop 压缩编码/解码器
DEFLATE	Org.apache.hadoop.io.compress.DefaultCodec
Gzip	Org.apache.hadoop.io.compress.GzipCodec
bzip2	Org.apache.hadoop.io.compress.BZip2Codec
LZO	Com.hadoop.compression.lzo.LzopCodec

```
% hadoop FileDecompressor file.gz
```

属性名	类型	默认值	描述
io.compression.codecs	逗号分隔的类名	org.apache.hadoop.io.compress.DefaultCodec, org.apache.hadoop.io.compress.GzipCodec, org.apache.hadoop.io.compress.Bzip2Codec	用于压缩/解压的 CompressionCodec 的列表

序列化

- Writable

```
package org.apache.hadoop.io;

import java.io.DataOutput;
import java.io.DataInput;
import java.io.IOException;

public interface Writable {
  void write(DataOutput out) throws IOException;
  void readFields(DataInput in) throws IOException;
}
```

- Writeable
    1. Text 类
    2. 自定义 Writeable 类

MapReduce 的应用开发

- API 的配置

```
Configuration conf = new Configuration();
conf.addResource("configuration-1.xml");
assertThat(conf.get("color"), is("yellow"));
assertThat(conf.getInt("size", 0), is(10));
assertThat(conf.get("breadth", "wide"), is("wide"));
```

后定义的属性会覆盖之前的属性标记为 final 不能被覆盖

```
% hadoop fs -conf conf/hadoop-localhost.xml -ls .
Found 2 items
drwxr-xr-x - tom supergroup 0 2009-04-08 10:32 /user/tom/input
drwxr-xr-x - tom supergroup 0 2009-04-08 13:09 /user/tom/output
```

- Hadoop 的辅助类

GenericOptionsParser 也允许用户设置个人自定义的属性，例如：

```
% hadoop ConfigurationPrinter -D color=yellow | grep color
color=yellow
```

-D 选项用来将键“color”设置成“yellow”。-D 选项的优先级比配置文件中的属性优先级高。这是十分有用的：可以把默认属性放入配置文件，需要时用-D 选项

```
public interface Tool extends Configurable {
  int run(String [] args) throws Exception;
}
```

表 5-1：GenericOptionsParser 和 ToolRunner 选项

选项	说明
-D *property=value*	将指定值赋给指定的 Hadoop 配置属性。可覆盖在配置中的任何默认属性或站点属性以及用-conf 选项设置的属性
-conf *filename* ...	将指定文件加入配置的资源列表。这是一种设置站点属性或同时配置多组属性的简便方法。
-fs *uri*	用指定的 URI 来设置默认的文件系统。这是 -D fs.default.name=uri 的快捷方式
-jt host:post	设置 jobtracker 的 host 和 port。这是 -Dmapred.job.tracker=host: port 的快捷方式
-files *file1,file2,...*	从本地的文件系统(或任何符合模式的文件系统)中复制指定的文件到 jobtracker 使用的共享文件系统中(通常为 HDFS)，让 MapReduce 程序能够在任务工作目录中得到上述文件(请参阅第 8 章，进一步了解用于复制文件到 tasktracker 机器上的分布式缓存机制)
-archives	从本地文件系统(或任何具有模式的文件系统)复制指定的归档到 jobtracker 使用的共享文件系统中(通常是 HDFS)，读取它们并让 MapReduce 程序能在任务工作目录中获得上述文件
-libjars *jar1,jar2,...*	从本地文件系统(或任何符合模式的文件系统)复制指定的 JAR 文件到 jobtracker 使用的共享文件系统中(通常是 HDFS)，并将它们加入到 MapReduce 任务的类路径中。这个选项适用于传送作业需要的 JAR 文件

在集群上运行

我们需要运行驱动程序来启动作业，使用-conf 选项来指定将要运行作业的集群(使用-fs 和-it 选项可以达到同样的效果)：

```
% hadoop jar job.jar v3.MaxTemperatureDriver -conf conf/hadoop-
cluster.xml \input/ncdc/all max-temp
```

将 utility 类应用于 parse 记录的 mapper 当作业数过多 10000 时后他们的 ID 为把作业前缀替换成任务前缀，再加一个后缀

这个作业产生的输出很少，所以很容易从 HDFS 中将其复制到开发机器。若需要得到源模式目录中的所有文件，并在本地文件系统上把它们合并成一个单独的文件，可使用 Hadoop fs 命令中的-getmerge 选项：

```
% hadoop fs -getmerge max-temp max-temp-local
% sort max-temp-local | tail
```

如果检索的输出文件很小，可以使用另一种方法，用-cat 选项将输出文件打印到控制台：

```
% hadoop fs -cat max-temp/*
```

- 通过 map 输出流来调试作业
  使用远程调试器
  首先将 keep.failed.task.files 中的配置属性设置为 true，
  接着需要运行一个特殊的任务运行器即 IsolationRunner，用前面保留的文件作为输入。登录到任务失败的节点，寻找任务尝试目录。它可能是在本地

```
% export HADOOP_OPTS="-agentlib:jdwp=transport=dt_socket,
server=y,suspend=y,\address=8000"
```

  suspend=y 选项让 JVM 在运行代码前先等待调试器连接。用以下命令启动 IsolationRunner：

```
% hadoop org.apache.hadoop.mapred.IsolationRunner ../job.xml
```

  下一步，设置断点，连接远程调试器(所有主流的 JAVA IDE 都支持远程调试，可参阅说明文档)，然后任务会在你的控制下运行。可以这样重新运行任务任意多次。幸运的话，可以找到并修改错误。
- MapReduce 的工作流
  将复杂的问题分解成 MapReduce
  运行独立的作业：
  对于线性的，一个接一个运行作业。
  更复杂的借助类库
    。

2020/6/25

MapReduce 工作原理请求

1. 检查输出说明
2. 对作业输入划分
3. 将资源输入划分
4. 准备执行

初始化
任务的分配
失败（尝试也可被杀死这个和失败时杀死时不同的，不计入运势尝试次数;）

1. Tasktracker 失败如果失败发送心跳停止或者很少，会从 tasktracker 池中移除
2. Jobtracker 失败 Hadoop 没有处理这种失败的机制

作业的调度

可选择调度器

1. Fair Scheduler 让每个用户公平的共享集群，即公平分配；支持抢占

Shuffle 和排序 map 输出传到 reducer 作为后者的输入即称为 shuffle。

每个 map 任务都有一个环形内存缓冲区超出会溢写，新建一个溢写文件，完成后会合并

表 6-1：map 端可调整的属性

属性名称	类型	默认值	说明
io.sort.mb	int	100	对 map 输出进行排序时所使用的内存缓冲区的大小，以兆字节为单位
io.sort.record.percent	float	0.05	保留的 io.sort.mb 比例，用来储存 map 输出的记录边界。剩余的空间用来储存 map 输出记录本身
io.sort.spill.percent	float	0.80	map 输出内存缓冲和记录边界索引，两者使用比例的阈值。达到此值，开始至磁盘的溢写过程。
io.sort.factor	int	10	排序文件时一次合并的最大流数。这个属性也在 reduce 中使用。将此默认值增加到 100 是非常常见的
min.num.spills.for.combine	int	3	运行 combiner 所需要的溢写文件的最小数量（如果已定义 combiner 的话）
mapred.compress.map.output	boolean	false	压缩 map 输出
mapred.map.output.compression.codec	Class name	org.apache.hadoop.io.compress.DefaultCodec	压缩 map 输出的编解码器
task tracker.http.threads	int	40	每个 tasktracker 用于将 map 输出传给 reducer 的工作线程的数量。这是集群范围的设置，不能由某个单独的作业进行设置

表 6-2：reduce 端可调整的属性

属性名称	类型	默认值	描述
mapred.reduce.parallel.copies	int	5	用于将 map 输出复制到 reducer 的线程的数量

续表

属性名称	类型	默认值	描述
mapred.reduce.copy.backoff	int	300	在声明失败之前，reducer 获取一个 map 输出所花费的最大时间，以秒为单位。如果失败，reducer 可以根据此时间尝试重传。（利用指数后退(exponential backoff)[2]）
io.sort.factor	int	10	排序文件时一次合并的流的最大数量。这个属性也在 map 端使用
mapred.job.shuffle.input.buffer.percent	float	0.70	整个堆空间的百分比，用于 shuffle 的复制阶段，分配给 map 输出缓存
mapred.job.shuffle.merge.percent	float	0.66	map 输出缓存（由 mapred.job.shuffle.input.buffer.percent 定义），其使用比例的阈值，用于启动合并输出和磁盘溢写的过程
mapred.inmem.merge.threshold	int	1000	启动合并输出和磁盘溢写过程的最大 map 输出数量。0 或者更小的数意味着没有阈值限制，并且溢写行为将由 mapred.job.shuffle.merge.percent 单独控制
mapred.job.reduce.input.buffer.percent	float	0.0	在 reduce 过程中，用来在内存中保存 map 输出的空间占整个堆空间的比例。reduce 阶段开始时，内存中的 map 输出大小不能大于这个值。默认情况下，在

推测式执行（优化拖后腿的任务）

属性名称	类型	默认值	说明
mapred.map.tasks.speculative.execution	boolean	true	在一个 map 任务运行缓慢时确定是否运行额外的 map 任务实例
mapred.reduce.tasks.speculative.execution	boolean	true	在一个 reduce 任务运行缓慢时确定是否运行额外的 reduce 任务实例

会减少集群效率

任务 JVM 重用

表 6-4：任务 JVM 重用的属性

属性名称	类型	默认值	说明
mapred.job.reuse.jvm.num.tasks	int	1	在一个 tasktracker 上一给定的作业的每个 JVM 可以运行的任务最大数。值-1 表示无限制，同一个 JVM 可以被该作业的所有任务使用

重用后不会在一个 JVM 中运行，而是单独的 JVM

跳过坏记录

任务执行环境

表 6-5：任务执行环境的属性

属性名称	类型	说明	示例
mapred.job.id	String	作业 ID。(详见 5.5 节，了解格式的描述)	job_200811201130_0004
mapred.tip.id	String	任务 ID	task_200811201130_0004_m_000003
mapred.task.id	String	Task attempt ID (而非任务 ID)	attempt_200811201130_0004_m_000003_0
mapred.teask.partition	int	作业中的任务的 ID	3
mapred.task.is.map	boolean	此任务是否是 map 任务	true

MapReduce 的类型与格式

Hadoop MapReduce 中的 map 和 reduce 函数遵循以下的形式：

```
map: (K1, V1) → list(K2, V2)
reduce: (K2, list(V2)) → list(K3, V3)
```

通常来说，map 函数输入的键/值(k1 和 v1)类型与用于输出的(k2 和 v2)不同。然而，reduce 函数的输入类型必须与 map 函数的输出类型相一致。同时，reduce 函数的输入类型与输出类型(k3 和 v3)又可能与上述两者都不同。以下的 Java 代码体现了这个规定：

表 7-1：MapReduce 类型配置

属性	JobConf 方法 set 方法	输入类型		中间类型		输出类型	
		K1	V1	K2	V2	K3	V3
用于配置类型的属性							
mapred.input.format.class	setInputFormat()	•	•				
mapred.mapoutput.key.class	setMapOutputKeyClass()			•			
Mapred.mapoutput.value.class	setMapOutputValueClass()				•		
mapred.output.key.class	setOutputKeyClass()					•	
mapred.output.value.class	setOutputValueClass()						•
必须与类型一致的属性							
mapred.mapper.class	setMapperClass()	•	•	•	•		
mapred.map.runner.class	setMapRunnerClass()	•	•	•	•		
mapred.combiner.class	setCombinerClass()			•	•		
mapred.partitioner.class	setPartitionerClass()			•	•		
mapred.output.key.comparator.class	SetOutputKeyComparatorClass()			•			
mapred.output.value.groupfn.class	SetOutputValueGroupingComparator()			•			
mapred.reducer.class	setReducerClass()			•	•	•	•
mapred.output.format.class	setOutputFormat()					•	•

表 7-2：流操作分隔符属性

属性名称	类型	默认值	描述
stream.map.input.field.separator	String	\t	将输入键/值字符串作为字节流传递到流 map 处理时使用的分隔符
stream.map.output.field.separator	String	\t	将来自流 map 处理的输出分割成键/值字符串时使用的 separator，这些键/值将给 map 输出使用
stream.num.map.output.key.fields	int	1	由 stream.map.output.field.separator 分割的字段数量，这些字段可看成时 map 输出键
stream.reduce.input.field.separator	String	\t	将输入键/值字符串作为字节流传递到流 reduce 处理时使用的分隔符
stream.reduce.output.field.separator	String	\t	将来自流 reduce 处理的输出分割成键/值字符串时使用的分隔符，这些键/值将给最终 reduce 输出使用
stream.num.reduce.output.key.fields	int	1	由 stream.reduce.output.field.separator 分割的字段数量，这些字段可看成 reduce 输出键

输入格式

输入分片和记录

输入分片在 Java 中用 `InputSplit` 表示(和所有提到过的类一样，它在 `org.apache.hadoop.mapred` 包中)。

```
public interface InputSplit extends Writable {

  long getLength() throws IOException;

  String[] getLocations() throws IOException;

}
```

避免分割

有很多方法可以保证输入文件不被分割。第一种(最简单但不怎么漂亮的)方法就是增加最小分片大小，将它设置成大于要处理的最大文件大小，当然也可以将它设置成 long.MAX_VALUE，也就是输入分片大小的上限。另外一种方法就是继承 FileInputFormat 实体类，并且重载 isSplitable()方法[9]，将它的返回值设置为 false。例如，以下就是一个不可分割的 FileInputFormat：

```
import org.apache.hadoop.fs.*;
import org.apache.hadoop.mapred.TextInputFormat;

public class NonSplittableTextInputFormat extends
TextInputFormat {
  @Override
  protected boolean isSplitable(FileSystem fs, Path file) {
    return false;
  }
}
```

表 7-6：文件输入分片的属性值

属性名称	类型	说明
map.input.file	String	正在处理的输入文件路径
map.input.start	long	分片开始处的字节偏移量
map.input.length	long	分片的长度(按字节)

文本输入

NLineInputFormat 一次收入多行文本

二进制输入 SequenceFileInputFormat

数据库格式的输入 DBInputFormat

输出格式

2020/6/26

MapReduce 特性

1. 计数器:

计数器是一个非常有用的途径，用于收集有关作业的统计数据，无论是对于质量控制，还是应用层面的统计数据。它还有助于对问题的诊断。如果想在 map 或 reduce 任务中添加一条日志记录信息，那么通常更好的方法看是否可以用一个计数器来记录特定情况的发生。对于大型分布式作业来说，计数器的值比日志记录输出更容易获得，同时还可以得到这种条件发生次数的记录，而这比从日志文件集得到该记录容易得多。

内置计数器:

计数器被与它们相关的任务所维护，并定期发送到 tasktracker，然后发送到 jobtracker，这样就能在全局范围内汇总这些计数器(第 6 章“进展和状态更新”有详细描述)。这些内置的作业计数器实际上是 jobtracker 维护的，因此它们不必通过网络发送，这点不同于其他计数器，包括用户自定义计数器等。

自定义 java 计数器:

MapReduce 允许用户代码定义一组计数器。它们在 mapper 或 reducer 中可根据需要递增。计数器被定义为 Java 枚举型，这可以使相互关联的计数器成组。一个作业可以定义任意数量的枚举类型，每个枚举型可以有任意数量的字段。枚举的名称就是该组的名称，枚举的字段就是计数器的名称。计数器是全局的：MapReduce 框架在作业的最后汇总所有 map 和 reduce 中的计数器信息来统计出总数。

动态计数器

易读的计数器名

获取计数器

除了通过 Web 界面和命令行(使用 `hadoop job-counter`)，还可以通过 Java API 获取计数器。你可以一边运行作业一边获取，不过更通常的做法是在作业运行结束

见 p258

用户自定义流计数器

MapReduce 流程序可以通过发送一条特殊的格式化行信息到标准错误流来增加计数器，这是在该情况下的一种控制方法。该行信息必须具有如下格式：

```
reporter:counter:group,counter,amount
```

2. 排序

对数据进行排序是 MapReduce 的核心。

准备

部分排序

全局排序

二次排序

MapReduce 框架在记录到达 reducer 之前对这些记录按键排序。然而对于任何特定的键，其对应的值没有排序。每次运行时值的顺序甚至都不固定，因为它们来自不同的 map 任务，每次运行时完成时间都不同。一般而言，MapReduce 的程序都编写为不依赖于值到达 reduce 函数的顺序。然而，在特定的方式下排序和分组键来排序值是可行的。

3. 联结

MapReduce 能够在大型数据集之间进行联接(join)，但是相当程度上需要从零开始编写代码做联接。相对于编写 MapReduce 程序，可以考虑使用一个较高层次的框架，如 Pig，Hive 或 Cascading，其中的联接操作是其实现的核心部分。

如何实现联接这取决于数据集的大小和它们的划分情况。如果一个数据集很大(气温记录)，但另一部分则是小到可以分发到集群中的每个节点(如气象站元数据)，那么连接可由一个 MapReduce 作业实现，该作业将每个气象站的记录放在一起(如按 station ID 部分排序)。mapper 或 reducer 使用这个较小的数据集来查找一个气象站的元数据，这样每个记录就能一并输出。见本章“二次数据分布”上的这种做法，在那里我们重点讨论了将数据分发到 tasktracker 的机制。

如果两个数据集都过于大而不能复制到集群中的每个节点，则可以使用 MapReduce，使用一个 map 端联接或 reduce 端联接来连接它们，一个简单的例子就是一个用户数据库和一些用户活动的日志记录(如访问日志记录)。对于一般的应用，将用户数据库(或日志记录)分发到 MapReduce 的所有节点是不可行的。

4. 次要数据的分布

次要数据(side data)可以被定义为一个作业为处理主要数据集所需要的额外的只读数据。目前的挑战是为所有 map 或 reduce 任务(遍布集群)提供方便快捷的方式来访问次要数据。

除了本节所述的分布机制，还可以在内存中的静态区域缓存次要数据，因此，在同一作业中连续运行在同一 tasktracker 的任务可以共享数据。第 6 章曾介绍了如何启

分布式缓存:

工作原理

启动作业后，Hadoop 将-files 和-archive 选项指定的文件复制到 jobtracker 的文件系统(通常是 HDFS)。然后，在任务运行前，tasktracker 从 jobtracker 文件系统复制文件到本地磁盘作为缓存，这样任务可以访问这些文件。从任务的角度来看，该文件就存放在那儿(而并不关心它们从 HDFS 来)。

该 tasktracker 还负责维护一个引用计数，这个引用计数记录了使用缓存中各文件的任务数。任务运行后，该文件的引用计数减少 1，当它达到零时，就能被删除。当缓存超过一定规模——默认为 10 GB 时，文件就被删除给新文件腾出空间。缓存的大小可以通过设置配置属性 `local.cache.size` 更改，大小以字节计算。

虽然这种设计并不保证来自同一作业(运行于同一个 tasktracker 上)的后续作业会找到缓存中的文件，但因为一个作业中的任务通常被安排在同样的时间运行，因此其他作业很可能没有机会运行从而导致原有任务的文件从缓存中被删除。

文件是在 tasktrackers 上的`${mapred.local.dir}/taskTracker/archive` 目录下本地化的。然而，应用程序不需要知道这一点，因为这些文件是从任务的工作目录通过符号来链接的。

5. MapReduce 的类库

可查询对应的 java 文档

Hadoop 集群搭建

1. 集群说明

说明是 2008 年后期运行 Hadoop 数据节点和 tasktracker 机器的典型选择方案：

中央处理器

两个四核英特尔 Xeon 2.0 GHz CPU

内存

8 GB ECC RAM[①]

存储器

4×TB SATA 磁盘

网络

千兆位以太网

集群要有多大呢？这个问题不会有确定的答案，但是 Hadoop 的好处在于你可以从一个小的集群(像 10 个节点)开始，并且随着存储器和计算机需求的扩大而扩大。在很多方面，一个更好的问题是：集群会成长多快？考虑存储容量会得到较好的答案。

对一个小的集群(大约 10 个节点)，名称节点和 jobtracker 运行在单个主节点上，通常是能接受的(只要对于名称节点的元数据至少有一个副本存储在远程的文件系统中)。随着集群和存储在 HDFS 中的文件数量的增加，名称节点需要更多的主存，所以名称节点和 jobtracker 会被移到不同的机器。

第二名称节点会同名称节点运行在同样的机器上，但是因为主存使用的原因(第二名称节点同第一名称节点有相同的主存需求)，它最好运行在一个不同的硬件片上，特别是对于更大的集群。(这个问题在本章的“Master node scenarios”有更多详细的讨论。)运行名称节点的机器一般是 64 位硬件来避免 32 位结构中 Java 3 GB 堆空间的限制。[①]

网络拓扑

机架感知

为了得到 Hadoop 最大的性能，配置 Hadoop 很重要，它包含网络拓扑。如果集群在单一机架上运行，就没有什么要做的了，因为这是默认的。然而，对多机架的集群，我们需要映射节点到机架上。通过映射，放置 MapReduce 任务在节点中时，Hadoop 将优先做机架内传输而不是机架外传输(有更多的带宽可用)。HDFS 能更智能地放置副本，在性能和适应力上权衡。

网络位置(如节点和机架)可以表示成一棵树，它反映了网络中位置之间的“距离”。名称节点在决定哪里存放块的副本时，会用到网络位置；当一个 map 任务被分配到一个 tasktracker 上运行时，jobtracker 节点会使用网络位置来确定作为 map 任务输入最近副本的位置。

2. 集群的安装和建立

安装 java

创建 hadoop 用户

安装 hadoop
测试安装
3. SSH 配置

Hadoop 控制脚本依靠 SSH 来执行集群范围内的操作，例如停止和开始所有集群中后台程序的脚本。注意，控制脚本是可选的——集群范围内的操作也可由其他机制执行(例如一个分布式的 shell)。
首先，在 hadoop 用户账户中通过键入如下命令来生成一个 RSA 密钥：

```
% ssh-keygen -t rsa -f ~/.ssh/id_rsa
```

接下来我们需要确认公共密钥在我们需要连接到的集群所有机器的~/.ssh/authorized_keys 文件中。如果 hadoop 用户的主目录是一个 NFS 文件系统，像前面描述的那样，通过键入如下命令可使密钥在整个集群中共享：

```
% cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

最后确定 ssh 代理是否运行来检测你是否可以通过从主控机(master)到工作机(worker)的 ssh 安全验证[1]，然后运行 ssh-add 来保存密码。这样一来，以后就不再需要输入密码通过工作机的 ssh 安全验证了。

4. hadoop 设置
配置管理
环境设置
重要的 Hadoop 后台程序属性

Hadoop 有一些难以理解的配置属性。在这一小节中，我们将关注所有真实工作着的集群中，那些必须定义的属性(或者至少明白为什么默认值是合适的)。这些属性是在 Hadoop 位置文件中设定的：*core-site.xml*，*hdfs-site.xml*，和 *mapred-site.xml*。例 9-1 展示了一个典型的文件集合的范例。注意，大多数变量定义为 final，是为了防止它们被作业配置覆盖。第 5 章的“配置 API”小节进一步介绍了如何写

Hadoop 后台程序地址和端口

Hadoop 后台程序一般都会运行一个为后台程序间通信的 RPC 服务器(表 9-5)和一个提供给人们使用的网页 HTTP 服务器(表 9-6)。每一个服务器都可以通过设置网络地址和端口号来配置。定义网络地址为 0.0.0.0 后，Hadoop 将绑定所有的地址到当前机器上。当然，可以指定一个单一的地址去绑定。端口号 0 的指示服务器在一个空闲的端口上启动：这一般不推荐，因为与设置此集群内的防火墙策略不协调。

其他属性

集群成员

服务级授权

缓冲大小

HDFS 块大小

备用存储空间

垃圾

任务内存限制

作业调度器

5. 安装后

一旦创建 Hadoop 集群并开始运行，我们需要给予用户权限使用它。这就需要为每一个用户创建一个主目录，并且对它设置权限许可：

```
% hadoop fs -mkdir /user/username
% hadoop fs -chown username:username /user/username
```

这时设置目录空间限制比较合适。下面给用户目录设置了一个 1 TB 的限制：

```
% hadoop dfsadmin -setSpaceQuota 1t /user/username
```

6. Hadoop 集群基准测试

为了得到最佳的结果，应在一个没有其他使用者的集群上运行基准测试程序。事实上，在它投入服务前，用户还没有使用时是最好的。一旦用户在集群上有了周期性调度的作业。就很难找到何时集群没被使用(除非对用户安排停工期)，所以应该在这发生之前运行基准程序测试直至满意为止。

Hadoop 基准测试

Hadoop 带有一些基准测试程序，可以最少的准备成本轻松运行。基准测试程序被打包在测试程序 JAR 文件中，通过无参数地调用 JAR 文件可以得到其列表：

```
% hadoop jar $HADOOP_INSTALL/hadoop-*-test.jar
```

无参数地调用时，大多数基准测试程序都会显示使用说明。例如：

用 TestDFSIO 基准测试 HDFS

TestDFSIO 用来测试 HDFS 的 I/O 性能。它通过使用 MapReduce 作业来完成测试作为并行读写文件的便捷方法。每个文件的读或写都在单独的 map 任务中进行，并且 map 的输出可以用来收集统计刚刚处理过的文件。这个统计数据在 reduce 中累加起来得出一个汇总。

用排序测试 MapReduce

Hadoop 自带一个部分排序的程序。这对测试整个 MapReduce 系统很有用，因为整个输入数据集都会通过洗牌传输(至 reducer)。一共三个步骤：生成一些随机的数据，执行排序，然后验证结果。

其他基准测试

其实还有很多更多的 Hadoop 基准测试，但下面这些是广泛使用的。

- MRBench(用 mrbench 调用)多次运行一道小作业。它是排序测试的好搭档，检测小的作业是否运行良好。[①]
- NNBench(用 nnbench 调用)对名称节点的压力测试很有用。
- Gridmix 是一套设计为模拟一个现实的集群作业负荷的基准测试，可以模拟许多在实践中可见的数据存取模式。详情请见分发包中的 src/benchmarks/gridmix 目录。[②]

7. 云计算中的 Hadoop

尽管许多组织选择在内部运行 Hadoop，但使用云计算，在一个租用的硬件上或者作为一个服务运行 Hadoop 也是很流行的。比如，Cloudera 为 Hadoop 在公有或者私有云提供了工具(见附录 B)。

Amazon EC2 中的 Hadoop

设置

运行集群

```
% bin/hadoop-ec2 launch-cluster test-hadoop-cluster 5
```

运行 MapReduce 作业

作业需要在 EC2 中运行(EC2 主机名需被正确解析)，为此，我们需要传输作业的 JAR 文件给集群。脚本提供了一个方便的方法。下面的命令复制 JAR 文件到主节点：

```
% bin/hadoop-ec2 push test-hadoop-cluster /Users/tom/htdg-
examples/job.jar
```

这也是登录集群的捷径。这个命令可以登录主机(使用 SSH)，也可以通过 EC2 实例 ID 来登录集群中的任意节点。(还有一个 `screen` 指令的捷径，或许更好。)

```
% bin/hadoop-ec2 login test-hadoop-cluster
```

集群的文件系统是空的，所以在我们运行任务之前，需要向其中放入数据。通过使用 Hadoop 的 distcp 工具从 S3 中执行并行复制是传输数据到 HDFS 的有效方法：

```
# hadoop distcp s3n://hadoopbook/ncdc/all input/ncdc/all
```

数据复制之后，我们可以用常用方法运行一道作业：

```
# hadoop jar job.jar MaxTemperatureWithCombiner input/ncdc/all
output
```

当然，我们也可以指定输入为 S3，效果相同。若在相同的输入数据上运行多道作业，最好先复制数据到 HDFS 上以节省带宽：

```
# hadoop jar job.jar MaxTemperatureWithCombiner
s3n://hadoopbook/ncdc/all output
```

我们可以通过使用 jobtracker 的网页 UI 跟踪作业的进度，见 *http://master_host:50030/*。

终止集群

要关闭集群，首先从 EC2 节点中退出/注销并且从我们的工作区发出 terminate-cluster 命令：

```
# exit
% bin/hadoop-ec2 terminate-cluster test-hadoop-cluster
```

此时要求确认是否关闭集群中的所有实例。