四 Hadoop 数据压缩

4.1 概述

压缩技术能够有效减少底层存储系统（HDFS）读写字节数。压缩提高了网络带宽和磁盘空间的效率。在 Hadoop 下，尤其是数据规模很大和工作负载密集的情况下，使用数据压缩显得非常重要。在这种情况下，I/O 操作和网络数据传输要花大量的时间。还有，Shuffle 与 Merge 过程同样也面临着巨大的 I/O 压力。

鉴于磁盘 I/O 和网络带宽是 Hadoop 的宝贵资源，数据压缩对于节省资源、最小化磁盘 I/O 和网络传输非常有帮助。不过，尽管压缩与解压操作的 CPU 开销不高，其性能的提升和资源的节省并非没有代价。

如果磁盘 I/O 和网络带宽影响了 MapReduce 作业性能，在任意 MapReduce 阶段启用压缩都可以改善端到端处理时间并减少 I/O 和网络流量。

压缩 **Mapreduce 的一种优化策略：通过压缩编码对 Mapper 或者 Reducer 的输出进行压缩，以减少磁盘 IO，**提高 MR 程序运行速度（但相应增加了 cpu 运算负担）。

注意：压缩特性运用得当能提高性能，但运用不当也可能降低性能。

基本原则：

（1）运算密集型的 job，少用压缩

（2）IO 密集型的 job，多用压缩

4.2 MR 支持的压缩编码

压缩格式	hadoop 自带？	算法	文件扩展名	是否可切分	换成压缩格式后，原来的程序是否需要修改
DEFAULT	是，直接使用	DEFAULT	.deflate	否	和文本处理一样，不需要修改
Gzip	是，直接使用	DEFAULT	.gz	否	和文本处理一样，不需要修改
bzip2	是，直接使用	bzip2	.bz2	是	和文本处理一样，不需要修改
LZO	否，需要安装	LZO	.lzo	是	需要建索引，还需要指定输入格式
Snappy	否，需要安装	Snappy	.snappy	否	和文本处理一样，不需要修改

为了支持多种压缩/解压缩算法，Hadoop 引入了编码/解码器，如下表所示

压缩格式	对应的编码/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
Snappy	org.apache.hadoop.io.compress.SnappyCodec

压缩性能的比较

压缩算法	原始文件大小	压缩文件大小	压缩速度	解压速度
gzip	8.3GB	1.8GB	17.5MB/s	58MB/s
bzip2	8.3GB	1.1GB	2.4MB/s	9.5MB/s
LZO	8.3GB	2.9GB	49.3MB/s	74.6MB/s

http://google.github.io/snappy/

On a single core of a Core i7 processor in 64-bit mode, Snappy compresses at about 250 MB/sec or more and decompresses at about 500 MB/sec or more.

4.3 压缩方式选择

4.3.1 Gzip 压缩

优点：压缩率比较高，而且压缩/解压速度也比较快；hadoop 本身支持，在应用中处理 gzip 格式的文件就和直接处理文本一样；大部分 linux 系统都自带 gzip 命令，使用方便。

缺点：不支持 split。

应用场景：当每个文件压缩之后在 130M 以内的（1 个块大小内），都可以考虑用 gzip 压缩格式。例如说一天或者一个小时的日志压缩成一个 gzip 文件，运行 mapreduce 程序的时候通过多个 gzip 文件达到并发。hive 程序，streaming 程序，和 java 写的 mapreduce 程序完全和文本处理一样，压缩之后原来的程序不需要做任何修改。

4.3.2 Bzip2 压缩

优点：支持 split；具有很高的压缩率，比 gzip 压缩率都高；hadoop 本身支持，但不支持 native；在 linux 系统下自带 bzip2 命令，使用方便。

缺点：压缩/解压速度慢；不支持 native。

应用场景：适合对速度要求不高，但需要较高的压缩率的时候，可以作为 mapreduce 作业的输出格式；或者输出之后的数据比较大，处理之后的数据需要压缩存档减少磁盘空间并且以后数据用得比较少的情况；或者对单个很大的文本文件想压缩减少存储空间，同时又需要支持 split，而且兼容之前的应用程序（即应用程序不需要修改）的情况。

4.3.3 Lzo 压缩

优点：压缩/解压速度也比较快，合理的压缩率；支持 split，是 hadoop 中最流行的压缩格式；可以在 linux 系统下安装 lzop 命令，使用方便。

缺点：压缩率比 gzip 要低一些；hadoop 本身不支持，需要安装；在应用中对 lzo 格式的文件需要做一些特殊处理（为了支持 split 需要建索引，还需要指定 inputformat 为 lzo 格式）。

应用场景：一个很大的文本文件，压缩之后还大于 200M 以上的可以考虑，而且单个文件越大，lzo 优点越越明显。

4.3.4 Snappy 压缩

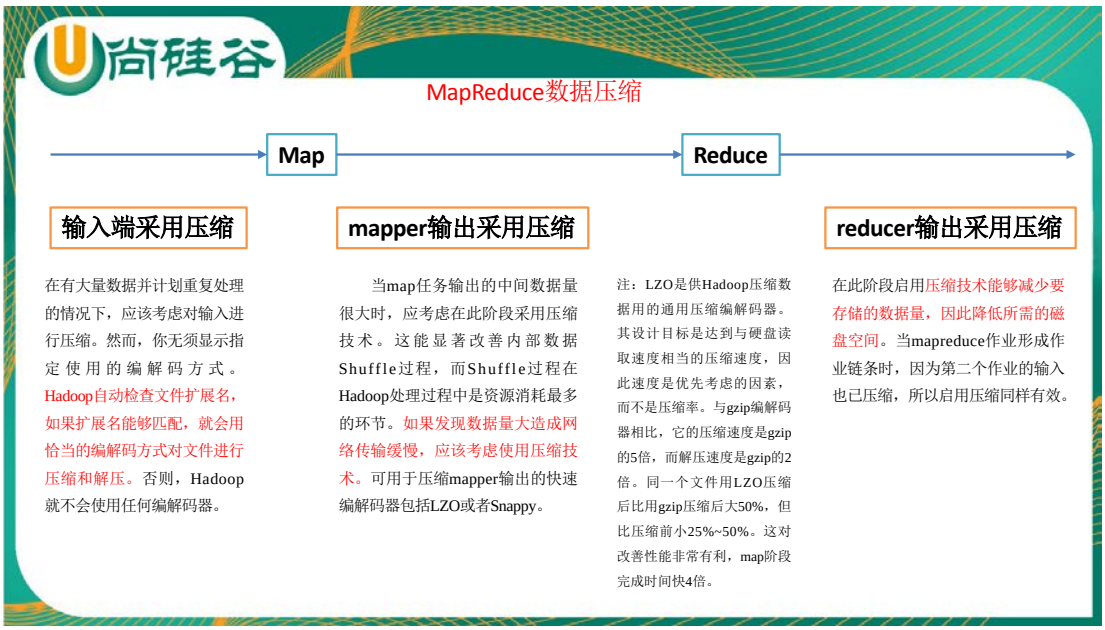
优点：高速压缩速度和合理的压缩率。

缺点：不支持 split；压缩率比 gzip 要低；hadoop 本身不支持，需要安装；

应用场景：当 Mapreduce 作业的 Map 输出的数据比较大的时候，作为 Map 到 Reduce 的中间数据的压缩格式；或者作为一个 Mapreduce 作业的输出和另外一个 Mapreduce 作业的输入。

4.4 压缩位置选择

压缩可以在 MapReduce 作用的任意阶段启用。



4.5 压缩参数配置

要在 Hadoop 中启用压缩，可以配置如下参数：

参数	默认值	阶段	建议
io.compression.codecs（在 core-site.xml 中配置）	org.apache.hadoop.io.compress.DefaultCodec, org.apache.hadoop.io.compress.GzipCodec,	输入压缩	Hadoop 使用文件扩展名判

	org.apache.hadoop.io.compress.BZip2Codec		断是否支持某种编解码器
mapreduce.map.output.compress（在 mapred-site.xml 中配置）	false	mapper 输出	这个参数设为 true 启用压缩
mapreduce.map.output.compress.codec（在 mapred-site.xml 中配置）	org.apache.hadoop.io.compress.DefaultCodec	mapper 输出	使用 LZO 或 snappy 编解码器在此阶段压缩数据
mapreduce.output.fileoutputformat.compress（在 mapred-site.xml 中配置）	false	reducer 输出	这个参数设为 true 启用压缩
mapreduce.output.fileoutputformat.compress.codec（在 mapred-site.xml 中配置）	org.apache.hadoop.io.compress.DefaultCodec	reducer 输出	使用标准工具或者编解码器，如 gzip 和 bzip2
mapreduce.output.fileoutputformat.compress.type（在 mapred-site.xml 中配置）	RECORD	reducer 输出	SequenceFile 输出使用的压缩类型：NONE 和 BLOCK

4.6 压缩实操案例

4.6.1 数据流的压缩和解压缩

CompressionCodec 有两个方法可以用于轻松地压缩或解压缩数据。要想对正在被写入一个输出流的数据进行压缩，我们可以使用 createOutputStream(OutputStreamout)方法创建一个 CompressionOutputStream，将其以压缩格式写入底层的流。相反，要想对从输入流读取

而来的数据进行解压缩，则调用 createInputStream(InputStreamin)函数，从而获得一个CompressionInputStream，从而从底层的流读取未压缩的数据。

测试一下如下压缩方式：

DEFLATE	org.apache.hadoop.io.compress.DefaultCodec
gzip	org.apache.hadoop.io.compress.GzipCodec
bzip2	org.apache.hadoop.io.compress.BZip2Codec

```
package com.atguigu.mapreduce.compress;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IOUtils;
import org.apache.hadoop.io.compress.CompressionCodec;
import org.apache.hadoop.io.compress.CompressionCodecFactory;
import org.apache.hadoop.io.compress.CompressionInputStream;
import org.apache.hadoop.io.compress.CompressionOutputStream;
import org.apache.hadoop.util.ReflectionUtils;

public class TestCompress {

	public static void main(String[] args) throws Exception {
		compress("e:/hello.txt","org.apache.hadoop.io.compress.BZip2Codec");
//		decompress("e:/hello.txt.bz2");
	}

	// 压缩
	private static void compress(String filename, String method) throws Exception {

		// 1 获取输入流
		FileInputStream fis = new FileInputStream(new File(filename));

		Class codecClass = Class.forName(method);

		CompressionCodec codec = (CompressionCodec) ReflectionUtils.newInstance(codecClass, new Configuration());

		// 2 获取输出流
		FileOutputStream fos = new FileOutputStream(new File(filename
```

```
+codec.getDefaultExtension()));
            CompressionOutputStream cos = codec.createOutputStream(fos);

            // 3 流的对拷
            IOUtils.copyBytes(fis, cos, 1024*1024*5, false);

            // 4 关闭资源
            fis.close();
            cos.close();
            fos.close();
        }

        // 解压缩
        private static void decompress(String filename) throws FileNotFoundException, IOException {

            // 0 校验是否能解压缩
            CompressionCodecFactory factory = new CompressionCodecFactory(new Configuration());
            CompressionCodec codec = factory.getCodec(new Path(filename));

            if (codec == null) {
                System.out.println("cannot find codec for file " + filename);
                return;
            }

            // 1 获取输入流
            CompressionInputStream cis = codec.createInputStream(new FileInputStream(new File(filename)));

            // 2 获取输出流
            FileOutputStream fos = new FileOutputStream(new File(filename + ".decoded"));

            // 3 流的对拷
            IOUtils.copyBytes(cis, fos, 1024*1024*5, false);

            // 4 关闭资源
            cis.close();
            fos.close();
        }
}
```

4.6.2 Map 输出端采用压缩

即使你的 MapReduce 的输入输出文件都是未压缩的文件，你仍然可以对 map 任务的中

间结果输出做压缩，因为它要写在硬盘并且通过网络传输到 reduce 节点，对其压缩可以提高很多性能，这些工作只要设置两个属性即可，我们来看下代码怎么设置：

1）给大家提供的 hadoop 源码支持的压缩格式有：BZip2Codec 、DefaultCodec

```
package com.atguigu.mapreduce.compress;
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.compress.BZip2Codec;
import org.apache.hadoop.io.compress.CompressionCodec;
import org.apache.hadoop.io.compress.GzipCodec;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCountDriver {

    public static void main(String[] args) throws IOException, ClassNotFoundException, InterruptedException {

        Configuration configuration = new Configuration();

        // 开启 map 端输出压缩
        configuration.setBoolean("mapreduce.map.output.compress", true);
        // 设置 map 端输出压缩方式
        configuration.setClass("mapreduce.map.output.compress.codec", BZip2Codec.class, CompressionCodec.class);

        Job job = Job.getInstance(configuration);

        job.setJarByClass(WordCountDriver.class);

        job.setMapperClass(WordCountMapper.class);
        job.setReducerClass(WordCountReducer.class);

        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(IntWritable.class);

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

        FileInputFormat.setInputPaths(job, new Path(args[0]));
```

```
            FileOutputFormat.setOutputPath(job, new Path(args[1]));

            boolean result = job.waitForCompletion(true);

            System.exit(result ? 1 : 0);
        }
}
```

2）Mapper 保持不变

```
package com.atguigu.mapreduce.compress;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class WordCountMapper extends Mapper<LongWritable, Text, Text, IntWritable>{

    @Override
    protected void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
        // 1 获取一行
        String line = value.toString();
        // 2 切割
        String[] words = line.split(" ");
        // 3 循环写出
        for(String word:words){
            context.write(new Text(word), new IntWritable(1));
        }
    }
}
```

3）Reducer 保持不变

```
package com.atguigu.mapreduce.compress;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class WordCountReducer extends Reducer<Text, IntWritable, Text, IntWritable>{

    @Override
    protected void reduce(Text key, Iterable<IntWritable> values,
            Context context) throws IOException, InterruptedException {
```

```
            int count = 0;
            // 1 汇总
            for(IntWritable value:values){
                count += value.get();
            }

            // 2 输出
            context.write(key, new IntWritable(count));
        }
}
```

4.6.3 Reduce 输出端采用压缩

基于 workcount 案例处理

1）修改驱动

```
package com.atguigu.mapreduce.compress;
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.compress.BZip2Codec;
import org.apache.hadoop.io.compress.DefaultCodec;
import org.apache.hadoop.io.compress.GzipCodec;
import org.apache.hadoop.io.compress.Lz4Codec;
import org.apache.hadoop.io.compress.SnappyCodec;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCountDriver {

    public static void main(String[] args) throws IOException, ClassNotFoundException, InterruptedException {

            Configuration configuration = new Configuration();

            Job job = Job.getInstance(configuration);

            job.setJarByClass(WordCountDriver.class);

            job.setMapperClass(WordCountMapper.class);
            job.setReducerClass(WordCountReducer.class);
```

```
            job.setMapOutputKeyClass(Text.class);
            job.setMapOutputValueClass(IntWritable.class);

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

            FileInputFormat.setInputPaths(job, new Path(args[0]));
            FileOutputFormat.setOutputPath(job, new Path(args[1]));

            // 设置 reduce 端输出压缩开启
            FileOutputFormat.setCompressOutput(job, true);

            // 设置压缩的方式
            FileOutputFormat.setOutputCompressorClass(job, BZip2Codec.class);
//          FileOutputFormat.setOutputCompressorClass(job, GzipCodec.class);
//          FileOutputFormat.setOutputCompressorClass(job, DefaultCodec.class);

            boolean result = job.waitForCompletion(true);

            System.exit(result?1:0);
        }
}
```

2）Mapper 和 Reducer 保持不变（详见 4.6.2）