## Introduction

When using Hadoop, there are many challenges in dealing with large data sets. Regardless of whether you store your data in HDFS or in ASV[[1]](#footnote-1) the fundamental challenge is that large data volumes can easily cause I/O and network bottlenecks.

In this paper, we will describe the problem of data volumes and how we used compression to mitigate problems in large Hadoop jobs. We will discuss the use of compression in each phase of a MapReduce job, assess the compression tools and techniques that are available, and present a summary of our test results. We also describe how to enable compression and decompression using both command-line arguments and configuration files.

### Why Use Compression?

For data intensive workloads, I/O operation and network data transfer will take considerable time to complete. Moreover, the internal MapReduce “shuffle” process is also under huge I/O pressure, and must often “spill out” intermediate data to local disks before processing can advance from the Map phase to the Reduce phase.

Disk I/O and network bandwidth is a precious resource for any Hadoop cluster. Therefore, compressing files for storage can not only save disk space, but also speed up data transfer across the network. More importantly, when processing large volumes of data in a MapReduce job, the combination of data compression and decreased network load can sometimes bring significant performance improvements, due to the reduced I/O and network resource consumption throughout the MapReduce process pipeline.

### The Compression Trade-off

Enabling compression is typically a tradeoff between I/O and speed of computation. Compression will typically reduce I/O and decrease network usage. Compression can happen when the MapReduce code reads the data or when it writes it out.

When a MapReduce job is run against compressed data, CPU utilization is increased, because data must be de-compressed before files can be processed by the Map and Reduce tasks. Therefore, decompression usually increases the time of the job. However, we have found that, in most cases, the overall job performance will be improved by enabling compression in multiple phases of the job configuration.

### Compression Formats Supported by Hadoop

Hadoop supports multiple compression formats for both input and output. A compression format is commonly referred to as a *codec*, which is short for coder-decoder. In Hadoop, a codec is a set of compiled, ready to use Java libraries that a developer can invoke programmatically to perform data compression and decompression in a MapReduce job.

Each codec encapsulates an implementation of one algorithm for compression and decompression. Because each method of compression is based on a different algorithm (for reducing white space, or hashing characters, etc.), each codec also has different characteristics.

There are many compression formats available as Open Source software or as part of commercial tools. Some compression formats can be generated by multiple tools. For example, the ZIP format is supported by multiple tools, such as BZIP and ZIP.

Some compression formats are **splittable**, which can enhance performance when reading and processing large compressed files. Typically, a single large compressed file is stored in HDFS with many data blocks that are distributed across many data nodes. When the file has been compressed by using one of the splittable algorithms, the data blocks can be decompressed in parallel by multiple MapReduce tasks. However, if a file has been compressed by a non-splittable algorithm, Hadoop must pull all the blocks together and use a single MapReduce task to decompress them.

The standard distribution of Hadoop provides several built-in codecs, including Deflate, Gzip and Bzip2, LZ4, and Snappy.

The following table lists the built-in codecs that are provided with the Apache Hadoop distribution and with HDInsight, according to our tests. At the time of this writing (January 2013), not all the codecs supported by Hadoop were available with HDInsight. The following information is valid for the HDInsight 0.3 release. We recommend that you check the HDInsight site for up-to-date information about features added in the HDInsight 4.0 release.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Format** | **Codec** | **Extension** | **Splittable** | **Hadoop** | **HDInsight** |
| DEFLATE | org.apache.hadoop.io.compress.DefaultCodec | .deflate | N | Y | Y |
| Gzip | org.apache.hadoop.io.compress.GzipCodec | .gz | N | Y | Y |
| Bzip2 | org.apache.hadoop.io.compress.BZip2Codec | .bz2 | Y | Y | Y |
| LZO | com.hadoop.compression.lzo.LzopCodec | .lzo | N | Y | N |
| LZ4 | org.apache.hadoop.io.compress.Lz4Codec | .Lz4 | N | Y | N |
| Snappy | org.apache.hadoop.io.compress.SnappyCodec | .Snappy | N | Y | N |

**Note**: LZO has been removed from Hadoop in version 0.20+, because the LZO libraries are licensed under the GNU General Public License (GPL). If you need to use LZO, the format is still supported, so you can download the codec separately and enable it manually in your Hadoop cluster.

The question of which codec to use depends on your requirements. Each codec has advantages and disadvantages, which are explained very well by Tom White in his book, *Hadoop: The Definitive Guide*.

“All compression algorithms exhibit a space/time trade-off: faster compression and decompression speeds usually come at the expense of smaller space savings. The tools [for compression in Hadoop …] give some control over this trade-off at compression time by offering nine different options: –1 means optimize for speed, and -9 means optimize for space. For example, the following command creates a compressed file file.gz using the fastest compression method:

gzip -1 file

The different tools have very different compression characteristics. Gzip is a general-purpose compressor and sits in the middle of the space/time trade-off. Bzip2 compresses more effectively than gzip, but is slower. Bzip2’s decompression speed is faster than its compression speed, but it is still slower than the other formats. LZO, LZ4. And Snappy, on the other hand, all optimize for speed and are around an order of magnitude faster than gzip, but compress less effectively. Snappy and LZ4 are also significantly faster than LZO for decompression.”

### Tools Used in this Paper

We tested compression using an early release of the HDInsight service, and the Apache distribution of Hadoop. While we were experimenting with compression, HDInsight had been released only as a Community Technical Preview (CTP). At that time, we discovered some difference between the two implementations: some of the compression features were not fully integrated yet, and some codecs were missing. However, as the product evolves, we expect that features which were missing might become available in later releases of HDInsight.

## How and When to Use Compression

Compression can be enabled during one or many phases of a MapReduce job, to reduce the I/O and network bandwidth. Compression can significantly improve the end-to-end processing time whenever I/O or network issues are the bottleneck in a MapReduce job.

In this section, we present our recommendations regarding the use of compression in each of the different phases of a MapReduce job.

### Compression of Input Files

You should consider enabling compression if your input source files are very large (for example, over 500 GB) and if you want to run MapReduce jobs repeatedly against the same input data. When you submit a MapReduce job against compressed data in HDFS, Hadoop will determine whether the source file is compressed by checking the file name extension, and if the file name has an appropriate extension, Hadoop will decompress it automatically using the appropriate codec. Therefore, users do not need to explicitly specify a codec in the MapReduce job.

However, if the file name extension does not follow naming conventions, Hadoop will not recognize the format and will not automatically decompress the file. Therefore, to enable self-detection and decompression, you must ensure that the file name extension matches the file name extensions supported by each codec.

For very large files, we strongly recommend use of a splittable compression algorithm for compressing the input files. The use of a splittable algorithm lets Hadoop spawn multiple Map tasks for parallel computation. If you do not use a splittable algorithm, Hadoop must use a single Map task to process the entire compressed file, which is very inefficient and non-scalable, which leads to extremely slow performance.

### Compression of Output Files in the Map Phase

In general, enabling compression during the Map output phase will significantly reduce the internal ‘shuffle’ time, and is particularly recommended if the task log indicates that data has been written to a large number of intermediate data partitions in the local hard drive – – a process called spilling. Shuffling is the most expensive and time consuming process in MapReduce, especially when there is a lot of data spilling.

To reduce the amount of data that is spilled to disk, you can use the following commands as part of your job configuration to specify that compression should be enabled on input data:

**Hadoop 0.20 and earlier**

Mapred.compress.map.output=true;

Mapred.map.output.compression.codec={Codec}

**Hadoop 0.21 and later**

mapreduce.map.output.compress

mapreduce.map.output.compress.codec={Codec}

For more information about the details of the MapReduce shuffle process, we recommend that you read Tom White’s book, *Hadoop: The Definitive Guide*.

### Compression of Final Job Output

Enabling compression on the final outputs of a MapReduce job not only reduces the amount of storage required, but also makes it faster to write the output file. This secondary benefit is particularly valuable when you chain MapReduce jobs together. When compression is enabled on the output of the first job, you not only reduce the amount of storage required, but also increase the performance of the second job in the chain, because the input file is already compressed by the first job. Moreover, compression of output data is useful for archiving data.

To enable compression of a MapReduce job’s outputs, you specify parameters such as the following:

**Hadoop 0.20**

mapred.output.compress=true;

mapred.map.output.compression.codec={Codec}

**Hadoop 0.23**

mapreduce.output.fileoutputformat.compress=true

mapreduce. output.fileoutputformat.compress.codec={Codec}

Note that the commands listed here have been tested with HDInsight.

The first set of parameters work well in the early CTP release of HDInsight we were using,. Note that the Hadoop version used in the CTP release of HDInsight is V1.1.0, which comes from the Hadoop 0.20 branch.

However, the commands used for Hadoop 0.23 do not work in HDInsight, because Hadoop versions 2.x are in fact based on a separate branch, renamed from the Hadoop 0.23 branch.

Therefore, generally speaking, settings that apply to Hadoop 0.23 do not work in HDInsight.

For more information about the different versions of Hadoop, see the Apache site, or this Wikipedia entry:

http://en.wikipedia.org/wiki/Apache\_Hadoop

For more information about implementations of Hadoop on windows, see the Hortonworks site:

http://hortonworks.com/partners/microsoft/

### Defining or Changing Compression Settings

There are two ways that you can define a compression algorithm as part of a MapReduce job:

* By setting parameters in the MapReduce job
* By using a global configuration

The following examples demonstrate how to do this using either method.

#### Defining compression for individual jobs

Compression settings can be applied in an individual MapReduce job by adding the argument *–D* and specifying arguments, or by using the *–conf* argument and adding the parameters in a configuration file.

-D <property=value>

-Dmapred.output.compress=true

-Dmapred.output.compression.codec=org.apache.hadoop.io.compress.BZip2Codec

To use these command line arguments, you must have enabled the Tool interface as part of the MapReduce job. For more information on the Tool interface, see <http://hadoop.apache.org/docs/r1.0.3/api/org/apache/hadoop/util/Tool.html>.

If you want to define the compressions method using a configuration file, use the following command-line argument:

-conf <configuration file>

The configuration file that you call to specify the compression method for the output file should adhere to the following format:

<?xml version="1.0" encoding="utf-8" ?>

<configuration>

<property>

<name>mapred.output.compress</name>

<value>true</value>

</property>

<property>

<name>mapred.output.compression.codec</name>

<value>org.apache.hadoop.io.compress.BZip2Codec</value>

</property>

</configuration>

#### Defining global compression settings

User can also specify compression settings in HDInsight configuration files (core-site.xml or hdfs-site.xml) to enable global compression that applies to all submitted MapReduce jobs. For an example, see the section, [Defining compression for individual jobs](#_Defining_compression_for).

**Important**: Compression settings that you add to the file, hdfs-site.xml, will take precedence over settings in the file, core-site.xml.

## Compression Methods

In this section we describe specific methods that you can use to compress files in the different phases of a MapReduce job. The following options are available:

* Use Open Source or commercial software for compression.
* Use .NET libraries with equivalent or similar functionality.
* Use the tools included with Hadoop.

Where possible, we have also provided a list of the command-line syntax options for each tool, for your convenience.

### Open Source or Commercial Compression Tools

Many Open Source and commercial tools have been developed for compression, and many of them are compatible with the Hadoop ecosystem. You just need to download the desired software, process the file locally, and then upload the compressed file into HDFS (the Hadoop file system) or ASV (the Azure storage system).

The tradeoffs between the tools are generally in cost, user familiarity, speed of compression and decompression, and the amount of compression. You can use whichever tool meets your needs, but we discuss a few of the popular alternatives here.

* BZIP2
* GZIP
* LZOP
* LZ4

#### Bzip2

Bzip is a popular tool which includes a version for Windows.

|  |  |
| --- | --- |
| Download site | <http://gnuwin32.sourceforge.net/packages/bzip2.htm> |
| Supported codecs | BZip2 |
| License | This library has been released under the GPL license. |
| Supported platforms | Apache Hadoop distribution  HDInsight |

Once you have downloaded the software and run the setup package to install the compression tools, you can find the tools in the following default location:

<drive>:\Program Files (86)\GnuWin32\bin folder.

##### Usage and parameters

Bzip2 [flags and input files in any order]

|  |  |
| --- | --- |
| **Flag** | **Description** |
| -h or -help | Print this message. |
| -d or -decompress | Force decompression. |
| -z or -compress | Force compression. |
| -k or -keep | Keep (don't delete) input files. |
| -f or -force | Overwrite existing output files. |
| -t or -test | Test compressed file integrity. |
| -c or -stdout | Output to standard output. |
| -q or -quiet | Suppress noncritical error messages. |
| -v or -verbose | Be verbose.  Use two -v arguments to get more detail. |
| -L or -license | Display software version and license. |
| -V or -version | Display software version and license. |
| -s or –small | Use less memory (maximum is 2500k). |
| -1 … -9 | Set block size to a value ranging from 100k to 900k |
| -fast | Alias for -1, meaning a block size of 100k. |
| -best | Alias for -9, meaning a block size of 900k. |

#### Gzip

The latest version of the suite includes both Bzip2.exe and Gzip.exe.

|  |  |
| --- | --- |
| Download site | <http://gnuwin32.sourceforge.net/packages/gzip.htm> |
| Supported codecs | **GZip** |
| License | This library has been released under the GPL license. |
| Supported platform | Apache Hadoop distribution  HDInsight |

##### Usage and parameters

Gzip [Option] [File]

By default, files are compressed in-place.

Arguments that are mandatory for long options are also mandatory for short options.

|  |  |
| --- | --- |
| **Flag** | **Description** |
| -a or -ascii | For ASCII text; convert end-of-line using the current locale. |
| -c or -stdout | Write to standard output and keep original files unchanged. |
| -d or -decompress | Decompress file. |
| -f or -force | Force overwrite of output file and compress links. |
| -h or -help | Display help for this command or argument. |
| -k or –keep | Keep (don't delete) input files. |
| -l or -list | list compressed file contents |
| -L or -license | display software license |
| -n or -no-name | Do not save or restore the original name and time stamp. |
| -N or -name | Save or restore the original name and time stamp. |
| -q or -quiet | Suppress all warnings. |
| -r or -recursive | Operate recursively on directories. |
| -S or -suffix | Use the suffix SUF on compressed files. |
| -t or -test | Test compressed file integrity. |
| -V or -version | Display version number. |
| -1 or -fast | Compress faster. |
| -9 or -best | Compress better. |
| -rsyncable | Make rsync-friendly archive. |

#### LZOP

**Lzop** is a file compression utility very similar to Bzip2. The LZO algorithm is optimized for speed and does not compress as much. This can be either an advantage or disadvantage, depending on whether you want speed or space.

|  |  |
| --- | --- |
| Download site | <http://www.lzop.org/> |
| Supported codes | LZO |
| License | LZOP is Open Source software but is copyrighted, and is distributed under the terms of GNU General Public License (GPL). |
| Supported platform | Apache Hadoop distribution only.  Depending on your version of Hadoop, you might need to download the LZOP codec should separately and enabled the codec manually in your Hadoop cluster. |

##### Usage and parameters

lzop [-dxlthIVL19] [-qvcfFnNPkUp] [-o file] [-S suffix] [file..]

|  |  |
| --- | --- |
| **Command** | **Description** |
| -1 | Compress faster. |
| -9 | Compress better. |
| -d | Decompress. |
| -x or -dPp | Extract. |
| -l | List compressed file. |
| -t | Test compressed file. |
| -I | Display system information. |
| -V | Display version number. |
| -h | Display help for this command or argument. |
| -L | Display software license. |
| **Option** | **Description** |
| -q | Be quiet (minimal error and status messages). |
| -v | Be verbose (display all status and error messages). |
| -c | Write to standard output (console). |
| -oFILE | Write output to specified file, ‘FILE’. |
| -p | Write output to the current directory. |
| -pDIR | Write output to the specified path 'DIR'. |
| -f | Force overwrite of output files. |
| -n | Do not restore the original file name. |
| -N | Restore the original file name (default). |
| -P | Restore or save the original path and file name. |
| -S.suf | Use the specified suffix, SUF, on compressed files. |
| -U | Delete input files after successful operation.  This behavior is the same as GZip and BZip2. |
| file… | File or list of files to decompress. If none given, try standard input. |

#### LZ4

LZ4 is a very fast and lossless compression algorithm.

If you are using Hadoop on Windows, you can also use the LZ4 compression algorithm through a command-line utility and a small Windows application. In our tests we used the command line tool and recommend it over the standalone application.

|  |  |
| --- | --- |
| Download site  Standalone Windows application  Command Line Utility for Windows | <http://www.lzop.org/>  <http://sd-1.archive-host.com/membres/up/182754578/LZ4_install_v13b.exe>  <http://sd-1.archive-host.com/membres/up/182754578/LZ4v13b.zip> |
| Supported codecs | LZ4 |
| License | LZ4 is currently included in the Hadoop distribution. |
| Supported platforms | Apache Hadoop distribution only |

##### Usage and parameters

LZ4.exe [arg] input [output]

Arguments can be appended. For example, the following syntax is valid:

LZ4.exe -c2t1s input [output]

|  |  |
| --- | --- |
| **Argument** | **Description** |
| -c# | Force compression at the level indicated by # (possible values are 0, 1, and 2). |
| -d | Force decompression. |
| -t# | Use the number of threads specified by # (default = number of cores)**.** |
| -b# | Benchmark files with the compression level specified by #. |
| -s | Silent mode (no stats animation). |
| -v | Verbose mode (force stats animation). |
| -o | Confirm destination overwrite. |
| -p | Pause at the end. |
| input | Can be 'stdin' (pipe) or a filename. |
| output | Can be 'stdout'(pipe) or a filename or 'null'. |

#### 7Zip

7Zip is a powerful compression tool that we highly recommend for use as a local compression processor. It is capable of maximizing computation resources on the local computer, and the user can configure the number of threads that can be used for file compression.

7zip provides several interfaces for interacting with the compression tools:

* Command line interface
* Graphical interface
* Microsoft windows shell integration

7zip has its own documentation, and it is easy to learn how to use the shell to process files. To view the 7ZIP documentation, open the directory where you installed 7zip, and look for the help documentation (typically provided as standalone document or text file.)

|  |  |
| --- | --- |
| Download site | <http://www.7-zip.org/download.html> |
| Supported codes | **BZip2, Gzip**  (7Zip does not provide LZO compression.) |
| License | 7Zip is open source software; most of the source code is under the GNU LGPL license. |
| Supported platforms | Apache Hadoop distribution  HDInsight |

##### Usage and parameters

Bzip2 [flags and input files in any order]

Gzip [Option] [File]

For a list of parameters or commands, see the sections for [BZip2](#_Bzip2) and [Gzip](#_Gzip_1).

##### Example

The following code sample demonstrates how to leverage the convenient 7zip Windows shell to perform bulk compression. The sample defines a command-prompt script (BAT file) that compresses multiple files in the specified folder.

You can customize the sample by changing the compression algorithm or other parameters, such as the file folder.

@echo off

set sPath=

set /p sPath=Please enter the source path(Such as D:\Hadoop):

echo.

set dPath=

set /p dPath=Please enter the destination path:

echo.

echo Waiting for traversing...

path=C:\Program Files\7-Zip;C:\Program Files (x86)\7-Zip2

for /f %%i in ('dir/s/b %sPath%\\*') do 7z a -tbzip2 %dPath%\"%%~ni%%~xi".bz2 "%%i" -r

echo Complete press........

pause

### .NET Compression Libraries

If any coding is needed as part of the Hadoop job workflow, consider using the Microsoft .NET Framework. There are .NET libraries that can easily generate compressed files, and most of the outputs of these libraries (with the exception of Deflate) are recognized by both HDFS and ASV.

This section provides a description of the .NET libraries that can be used for compression, together with code samples that illustrate how to use them to generate compressed files.

We also briefly discuss some Open Source libraries for the same purpose, and explain the use of the integrated compression tool shell available for.NET programmers.

#### System.IO.Compression

The .NET Framework (version 2.0 or newer) includes a built-in namespace, System.IO.Compression, that provides basic stream compression and decompression classes that can be used to compress files using various algorithms.

The following code sample demonstrates how to use these libraries to bulk compress source files in a specific folder, using the Gzip format.

string[] filenames = Directory.GetFiles(sourcePath);

foreach (string file in filenames)

{

string filePath = "";

if (destinationPath == string.Empty)

filePath = Path.GetFileName(file) + ".gz";

else

filePath = destinationPath + "\\" + Path.GetFileName(file) + ".gz";

using (FileStream fs = new FileStream(file, FileMode.Open, FileAccess.Read))

{

using (FileStream stream = new FileStream(destinationPath, FileMode.Create, System.IO.FileAccess.Write))

{

using (GZipStream gZipStream = new GZipStream(stream, CompressionMode.Compress))

{

byte[] buffer = new byte[fs.Length];

fs.Read(buffer, 0, buffer.Length);

fs.Seek(0, SeekOrigin.Begin);

gZipStream.Write(buffer, 0, buffer.Length);

gZipStream.Close();

stream.Close();

fs.Close();

}

}

}

}

### Open Source Libraries

There are also multiple Open Source libraries for use with the .NET Framework, many of which can be used to compress and decompress files. Of these, we had the time to test just one: the SharpZipLib .NET compression library.

#### SharpZipLib

**SharpZipLib** is very easy to use, and supports multiple compression algorithms.

|  |  |
| --- | --- |
| Download site | <http://sharpziplib.com/> |
| Supported codecs | Zip, GZIP, Tar, BZip2 |
| Supported platforms | Apache Hadoop distribution  HDInsight |
| License | This library has been released under the GPL license.  For more information, see:  <http://www.icsharpcode.net/OpenSource/SharpZipLib/> |

The following code snippets demonstrate some of the basic functions of this library:

##### Example: Create GZIP archives using SharpLibZip

GZipClass gZipClass = new GZipClass();

gZipClass.GZipFiles(inputPath, outputPath, out errorMsg);

##### Example: Create BZip2 archives using SharpLibZip

BZip2Class bZip2Class = new BZip2Class();

bZip2Class.BZip2Files(inputPath, outputPath, out errorMsg);

For detailed information and more demo code, see the following site

<http://www.icsharpcode.net/OpenSource/SharpZipLib/>

#### SnappySharp and LZ4Sharp

Kintaro-SnappySharp and LZ4Sharp are C# native implementations for the Snappy and LZ4 algorithms, which can be downloaded from GitHub.

|  |  |
| --- | --- |
| Download site | SnappySharp: <https://github.com/Kintaro/SnappySharp>  LZ4Sharp: <https://github.com/stangelandcl/LZ4Sharp> |
| Supported codecs | Snappy, LZ4 |
| Supported platforms | Apache Hadoop distribution |

However, the C# port of Snappy does not appear to be complete; As of the time of this writing, some methods in the code appear to be in progress.

#### Deflate

The .NET framework includes the DeflateStream class, which provides methods and properties for compressing and decompressing streams by using the Deflate algorithm. However, we were unable to generate Deflate files that can be decompressed by HDInsight using version of the.NET framework earlier than version 4.5. However, starting with the .NET Framework 4.5, the DeflateStream class uses the zlib library. As a result, it provides a better compression algorithm and, in most cases, a smaller compressed file than it provides in earlier versions of the .NET Framework. Other teams within Microsoft have reported that .NET 4.5 libraries worked for their HDInsight projects.

The following MSDN article provides the syntax for the class details and also a code sample.

<http://msdn.microsoft.com/EN-US/library/0y917bht(v=VS.110,d=hv.2).aspx>

HQL is another method that we have successfully tested for using the Deflate algorithm in HDInsight. For details please refer to section, [HIVE Queries for Compression](#_HIVE_Queries_for).

### Integrating Compression into .NET Programs

We have already described some of the Open Source tools and algorithms that can be used for compression and decompression. Among these are command-line tools that provide rich parameters for customizing the way that files are compressed and decompressed. Ideally, one would like to integrate these command-line tools into our .NET code.

We experimented with these tools and found that most of them perform adequately when called from .NET code, and have outputs that are supported by HDFS and ASV.

The following code sample demonstrates how you can easily invoke 7zip to compress a file.

##### How to: Use the 7z command-line utility in .NET code

1. In your solution, add a reference to 7z.exe in the project that performs compression.
2. Verify that the utility can write to the desired output folder.
3. Invoke the utility by using the Process class, as shown in the following code sample:

static void Main()

{

string sourceFile = "d:\testio.txt";

string targetFile = "d:\testio.txt.bz2";

ProcessStartInfo processStart = new ProcessStartInfo();

processStart.FileName = "7z.exe";

// Use 7-zip

// specify a=archive, -tbzip2=bzip2 and hide the 7zip window

processStart.Arguments = "a -tbzip2 \"" + targetFile + "\" \"" + sourceFile + "\"";

processStart.WindowStyle = ProcessWindowStyle.Hidden;

Process process = Process.Start(processStart);

process.WaitForExit();

}

### HIVE Queries for Compression

If the files that you need to compress are typically structured files, such as .CSV files, and the files are already stored in HDFS, it might be quicker to compress the files using Hive, rather than NET code or third-party tools.

The trick is to simply use the Hive INSERT OVERWIRTE SELECT FROM query to generate compressed output, using the following syntax:

INSERT OVERWRITE DIRECTORY <Output folder> SELECT \* FROM <InputTable>

After you enable compression for this query (details shown below), the command will create multiple Map tasks running in parallel to compress each block without invoking a Reduce task. The job will compress each file block locally and dump them to the configured output folder.

To understand how this works under the covers, let’s look at an example.

If the size of a source file stored in HDFS is 1 GB and the block size configuration is 256MB, the uncompressed file would be divided and stored as 4 blocks in HDFS. Potentially each block could be on a different data node, and by default Hadoop creates three copies of each block. Hence Hadoop will create four map tasks, to select records from each block, compress them, and then output the compressed records to the output folder as a part of the output file.

For this example, let’s assume that compression reduces each file to about 25% of its original size. Given the original block size, the compressed files output by the Hive command should be about 64MB each.

Therefore, this technique alone can provide considerable reduction in storage space. Moreover, you can modify the default HDFS block size to influence the size of the compressed file.

##### How to: Use HIVE to compress a file in HDFS

1. Create a table in Hive. For this sample, we used HQL to create a demo table.
2. Make sure the table schema matches the file that you want to compress. In the following demo the files to be compressed are stored in the following HDFS folder:

user/Demo/txt

1. Use the following HIVE command creates a Hive table that is associated with the input files:

CREATE external table if not exists DemoTable( ID string, GUID string, ServerDateTime string)

ROW FORMAT delimited fields terminated by '09'

Location '/demo/txt';

1. Type the following Hive commands to ensure that output files are generated in a compressed format:

set mapred.output.compress=true;

set hive.exec.compress.output=true;

set mapred.output.compression.codec={Codec}

INSERT OVERWRITE DIRECTORY ‘/Demo\_out’ select \* from demotable;

The command in the example specifies the codec used for compression. Although we used Gzip in the example, different codecs can be specified, depending on what type of compression you want.

The following HQL command creates multiple Map tasks running in-parallel, to compress each data block without a Reduce task being involved. Each map task will compress one data block individually and multiple compressed file parts will be created in the ‘Demo\_out’ folder.

INSERT OVERWRITE DIRECTORY ‘/Demo\_out’ SELECT \* FROM demotable

You can verify that compression is achieved by applying the same DemoTable schema to a different table (for example, ‘DemoTableAfterCompress’) and associating the new table with the output folder (/Demo\_out). The new table will contain exactly the same data rows as the original table (DemoTable), and you can still query the data using Hive, but the storage size is only about 25% of the original table.

### Container File Format Compression

A final method that should be mentioned, but which we have not tested, is using the container file format provided by Hadoop.

Users who wish to leverage container file compression for sequence files and map files, which are features supported by both Hadoop and by HDInsight, should refer to Tom White’s book *Hadoop: The Definitive Guide*.

### Avro and Protobuf

Although not technically compression methods, we should mention Avro and Protobuf. Oneapproach that internal teams have taken is to serialize the data using Avro and then compress the data them using Deflate. This approach reportedly provides both good performance and good compression ratios. For additional details, see this blog by Denny Lee:

<http://dennyglee.com/2013/03/12/using-avro-with-hdinsight-on-azure-at-343-industries/>

## Testing Compression in Hadoop on Azure

Microsoft HDInsight can help users create Hadoop clusters in Windows Azure. The self-service Web UI makes it very easy for a user to create a Hadoop cluster using the HDInsight Service on Azure, and to run Map/Reduce jobs on it.

At the time of this writing, HDInsight supports multiple underlying distributed storage formats:

* HDFS
* Azure Blob Storage via the Azure Storage Vault (ASV)
* Amazon S3 Blob Storage

In our experiment, we tested compression in Windows Azure on both HDFS and Azure Blob Storage. We did not use Amazon S3 Blob Storage.

### Testing Goals and Process

The goal of our experiment was to test various file formats—plain text, Bzip2, Gzip and Lz4 compressed files—in both the Azure Blob Store and in HDFS, to determine whether all formats are recognized by MapReduce jobs.

The test application we used was the Word Count sample, which is provided along with most Hadoop installations. This test application counts how often each word occurs in the sample input file.

1. First, we configured the two storage systems.
2. We then uploaded an identical set of files into each system.
3. We ran the tests, and obtained the execution times for each test from the job logs.

The tests are described in more detail in this section.

### Configuring Azure Blob Storage (ASV)

Azure provides different types of storage, according to your needs. For more information about Azure storage offerings, see this site:

<http://social.technet.microsoft.com/wiki/contents/articles/1674.data-storage-offerings-on-the-windows-azure-platform.aspx>

The Azure Storage Vault (ASV) is an extension to the HDFS file system’s extensibility mechanism for integrating with external data sources that lets Hadoop (including the HDInsight service) use Azure blob storage. ASV is the recommended storage method for Hadoop job, as it provides numerous advantages for Hadoop jobs: no constraints on file size, geo-location, low storage costs, the ability for Hadoop to address the storage directly, and management of security by Azure.

To configure Azure Blob storage, follow the instructions in the following article:

<http://blogs.msdn.com/b/avkashchauhan/archive/2012/01/13/setting-windows-azure-blob-storage-asv-as-data-source-directly-from-portal-at-hadoop-on-azure.aspx>

After you have configured ASV, you can upload to Azure Blob Storage the files that you have compressed using any of the methods described earlier in this paper. To manage blobs, queues, and tables, we recommend Azure Storage Explorer. It is one of several tools that can be used to manage Azure storage objects, and we used it in these tests. You can download the tool from the following site:

<http://azurestorageexplorer.codeplex.com/>

### Compression Settings and Results

We used the same source files to run the Word Count test. For this test, we did not customize the compression algorithms or change parameters, but used the default values and the default file name extensions for compressed files.

The following table shows the codecs we tested, and the compression achieved with each codec.

Table 1. Compression results

|  |  |
| --- | --- |
| Uncompressed file | 1 GB |
| BZip | 335 MB |
| GZip | 338 MB |
| LZ4 | 557 MB |

The test also demonstrated that all of the compressed file formats can be recognized and processed by the Word Count application without code changes, regardless of whether the files are stored in HDFS or in Azure Blob Storage.

We also compared processing times, as compression algorithms often are optimized for either speed or compression, but not for both.

The following table shows total processing time for the Word Count application for each file type, compressed and uncompressed.

Table 2. Processing speed on compressed and uncompressed files

|  |  |
| --- | --- |
| **File Type** | **MapReduce time** |
| Plain text | 629 seconds |
| Bzip2 | 648 seconds |
| GZip | 588 seconds |
| LZ4 | \* |

\* Our tests show that LZ4 and Snappy are currently not supported by HDInsight.

### Summary of Tests

Our test results shows that all of the compressed files that we tested can be recognized and processed well by the Word Count program without requiring code changes, regardless of whether the files are stored in HDFS or in the Azure Blob Store.

Our tests also showed that LZ4 provides the fastest compression speed, with Gzip in second place. Bzip2 provided the slowest compression and decompression. The MapReduce job for Word Count processed the GZip compressed file faster than the BZip2 file, and processing of a plain text file was the slowest.

## Conclusion

Our results showed that enabling compression in Hadoop is a good practice in general:

* It can save a lot of storage space.
* It speeds up MapReduce job performance significantly.
* Compression is easy to do on a per-job or global basis.

To enjoy the benefits of compression, all you really need to do is add a few configuration settings to compress both input files and output files.

There are many options to compress data for Hadoop jobs in HDInsight:

* Users can enable compression and decompression in MapReduce jobs or globally by adding configuration settings or specifying command line parameters.
* User can choose from various third-party and Open Source tools.
* You can also use NET libraries, Hive compression in queries, or compression provided by the Hadoop container file format.

According to our tests, the following compression algorithms work well with the current HDInsight release:

* Gzip
* BZip2
* Deflate with restrictions (Hive only)

LZO, LZ4 and Snappy are not supported yet for HDInsight, but this may change in future releases.Given that compressing files will significantly speed up Hadoop jobs and queries, we recommend that you examine and adopt one of these options. If you implement a standard compression policy across the entire Hadoop cluster, you can specify the compression configuration settings in the cluster configuration files rather than defining the compression method in the code itself, and thereby achieve more consistency in file processing and performance.

## References and Appendices

**[01]** Nolan, Carl. MSDN Blog: “Hadoop Streaming and Windows Azure Blog Storage”. Jan 7 2012.

<http://blogs.msdn.com/b/carlnol/archive/2012/01/07/hadoop-streaming-and-azure-blob-storage.aspx>

**[02]** Beresford, James. Blog: “Using Azure Blob Storage as a Data Source for Hadoop on Azure”. July 17, 2012.

http://www.bimonkey.com/2012/07/using-azure-blob-storage-as-a-data-source-for-hadoop-on-azure/

**[03]** MSDN Library. “System.IO.Compression Namespace”

http://msdn.microsoft.com/en-us/library/system.io.compression(v=vs.80).aspx

**[04]** MSDN Library: “Package Class”

http://msdn.microsoft.com/en-us/library/system.io.packaging.package.aspx

**[05]** White, Tom. *Hadoop: The Definitive Guide*, 2nd edition. O’Reilly Media, Inc., 2011.

### About the Authors

The Microsoft IT Big Data Program is the result of a collaboration between multiple enterprise services organizations working in partnership with Microsoft’s HDInsight product development team. The Enterprise Architecture (EA) team initially envisioned and incubated a series of internal Big Data solutions on HDInsight, which were operationalized by the Data and Decision Sciences Group (DDSG).

Together, these teams have been tasked with assessing the usefulness of Hadoop for corporate and client applications, and for providing a reliable, cost-effective platform for future Big Data solutions within Microsoft.

As part of our development activities, we have extensively researched the Hadoop offerings from Apache, and have used the HDInsight platform for Windows Azure and on-premise Windows deployments in production and test environments of all sizes.

We are providing these research reports to the community at-large so that others can benefit from our experience and make more efficient use of Hadoop for data-intensive applications.

### Acknowledgments

We would like to gratefully acknowledge our colleagues who read these papers and provided technical feedback and support: Carl Nolan, Brian Swan, Pedro Urbina Escos, Benjamin Guinebertière, and Cindy Gross.

1. HDFS is Hadoop’s native distributed file system. In HDInsight Service hosted on Windows Azure, you can use the default Azure blob storage and access data by using the ASV (Azure Storage Vault) extension. With ASV, data is always available in the blob store, so you don't have to reload it every time you create a cluster. ASV is the default storage system used by the HDInsight Service. [↑](#footnote-ref-1)