SequenceFile

Overview

SequenceFile is a flat file consisting of binary key/value pairs. It is extensively used in MapReduce as input/output formats. It is also worth noting that, internally, the temporary outputs of maps are stored using SequenceFile.

The SequenceFile provides a Writer, Reader and Sorter classes for writing, reading and sorting respectively.

There are 3 different SequenceFile formats:

1. Uncompressed key/value records. 2. Record compressed key/value records - only 'values' are compressed here. 3. Block compressed key/value records - both keys and values are collected in 'blocks' separately and compressed. The size of the 'block' is configurable.

The recommended way is to use the SequenceFile.createWriter methods to construct the 'preferred' writer implementation.

The SequenceFile.Reader acts as a bridge and can read any of the above SequenceFile formats.

SequenceFile Formats

This section describes the format for the latest **'version 6**' of SequenceFiles.

Essentially there are 3 different file formats for SequenceFiles depending on whether *compression* and *block compression* are active.

However all of the above formats share a common *header* (which is used by the SequenceFile.Reader to return the appropriate key/value pairs). The next section summarises the header:

SequenceFile Common Header

- version - A byte array: 3 bytes of magic header **'SEQ**', followed by 1 byte of actual version no. (e.g. SEQ4 or SEQ6)
- keyClassName - String
- valueClassName - String
- compression - A boolean which specifies if *compression* is turned on for keys/values in this file.
- blockCompression - A boolean which specifies if *block compression* is turned on for keys/values in this file.
- compressor class - The classname of the CompressionCodec which is used to compress/decompress keys and/or values in this SequenceFile (if compression is enabled).
- metadata - SequenceFile.Metadata for this file (key/value pairs)
- sync - A sync marker to denote end of the header.

All strings are serialized using Text.writeString api.

The formats for Uncompressed and RecordCompressed Writers are very similar:

Uncompressed & RecordCompressed Writer Format

- Header
- Record
    - Record length
    - Key length
    - Key
    - (Compressed?) Value
- A sync-marker every few k bytes or so.

The sync marker permits seeking to a random point in a file and then re-synchronizing input with record boundaries. This is required to be able to efficiently split large files for MapReduce processing.

The format for the BlockCompressedWriter is as follows:

BlockCompressed Writer Format

- Header
- Record *Block*
    - A sync-marker to help in seeking to a random point in the file and then seeking to next *record block*.
    - CompressedKeyLengthsBlockSize
    - CompressedKeyLengthsBlock
    - CompressedKeysBlockSize
    - CompressedKeysBlock
    - CompressedValueLengthsBlockSize
    - CompressedValueLengthsBlock
    - CompressedValuesBlockSize
    - CompressedValuesBlock
      The compressed blocks of *key lengths* and *value lengths* consist of the actual lengths of individual keys/values encoded in ZeroCompressedInteger format.