Item 379
git_comments:

1. * * Licensed to the Apache Software Foundation (ASF) under one * or more contributor license agreements. See the NOTICE file * distributed with this work for additional information * regarding copyright ownership. The ASF licenses this file * to you under the Apache License, Version 2.0 (the * "License"); you may not use this file except in compliance * with the License. You may obtain a copy of the License at * * http://www.apache.org/licenses/LICENSE-2.0 * * Unless required by applicable law or agreed to in writing, software * distributed under the License is distributed on an "AS IS" BASIS, * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. * See the License for the specific language governing permissions and * limitations under the License.
2. Default parameters
3. Read command-line options
4. * * Check whether the batch length varies depending on data.
5. * * Heap memory used by the batch.
6. Decompressors need buffers for each stream
7. Snappy decompressor uses a second buffer
8. ColumnReaders use lots of memory; free old memory first
9. Account for firstRowOfStripe.
10. If a string column is read, use stripe datalength as a memory estimate * because we don't know the dictionary size. Multiply by 2 because * a string column requires two buffers: * in the input stream and in the seekable input stream. * If no string column is read, estimate from the number of streams.
11. **comment:** Do we need even more memory to read the footer or the metadata?
    label: code-design
12. All columns
13. String column
14. Map column
15. Binary column
16. Int column
17. Struct column (with a List subcolumn)
18. List column

git_commits:

1. **summary:** Fixed ORC-21: Add dynamic memory usage estimation.
   message: Fixed ORC-21: Add dynamic memory usage estimation. fixes apache/orc#21 Signed-off-by: Owen O'Malley <omalley@apache.org>

github_issues:

github_issues_comments:

github_pulls:

1. **title:** ORC-47. Fix MacOS compilation warnings.
   body:

github_pulls_comments:

1. +1 Looks good.

github_pulls_reviews:

jira_issues:

1. **summary:** Add functionality to estimate memory footprint
   description: ORC library allocates multiple large buffers to read and materialize ORC files. For stability of applications that use the library, it may be desirable to have an estimate (preferably, a tight upper bound) of a memory footprint.

jira_issues_comments:

1. We use two components for reading ORC files: a reader and a reusable batch that we fill up with data. Since the end user should not know or worry about the internal workings of these components, both should be able to report how much memory that need. Proposed solution: add methods uint64_t Reader::memoryUse() and uint64_t ColumnVectorBatch::memoryUse() that return an exact value (or at least an upper bound estimate) of the memory footprint of the respective classes/subclasses.
2. GitHub user asandryh opened a pull request: https://github.com/apache/orc/pull/10 Fixed ORC-21: Add functionality to estimate dynamic memory requirements An upper bound on memory requirements is provided by two components: - Reader::memoryUse() returns an upper bound on its memory needs. It depends on the file and columns read. - ColumnBatch::memoryUse() returns an upper bound on its memory needs. It depends on the file, columns, and number of rows read. The new utility FileMemory.cc compares estimated and actual memory usage. You can merge this pull request into a Git repository by running: $ git pull https://github.com/asandryh/orc master Alternatively you can review and apply these changes as the patch at: https://github.com/apache/orc/pull/10.patch To close this pull request, make a commit to your master/trunk branch with (at least) the following in the commit message: This closes #10 ---- commit c31b3ed204ba01ad3835d963ec2eb8343fd2981e Author: Aliaksei Sandryhaila <aliaksei.sandryhaila@hp.com> Date: 2015-07-08T15:06:57Z Added a stream block size parameter to ReaderOptions to change the size of InputStreams' buffers. commit 7114d6e89b7f96762b07479b397298edb8412a8c Author: Aliaksei Sandryhaila <aliaksei.sandryhaila@hp.com> Date: 2015-07-08T15:14:18Z Corrected type casting. commit d40382662e9b564354160fa8a0d833ab5c699a08 Author: Aliaksei Sandryhaila <aliaksei.sandryhaila@hp.com> Date: 2015-07-24T20:17:26Z Revert "Corrected type casting." This reverts commit 7114d6e89b7f96762b07479b397298edb8412a8c. commit 5992cc95d63bbcab362e685e65d0748277c7582d Author: Aliaksei Sandryhaila <aliaksei.sandryhaila@hp.com> Date: 2015-07-24T20:17:47Z Revert "Added a stream block size parameter to ReaderOptions to change the size of InputStreams' buffers." This reverts commit c31b3ed204ba01ad3835d963ec2eb8343fd2981e. commit fe2714d535c1acaccf1bbf25715f2e1934d76fcb Author: Aliaksei Sandryhaila <aliaksei.sandryhaila@hp.com> Date: 2015-07-24T20:23:42Z Merge remote-tracking branch 'upstream/master' commit 51ec0491dc294101d377045b3665786cb3fe435d Author: Aliaksei Sandryhaila <aliaksei.sandryhaila@hp.com> Date: 2015-07-29T16:37:13Z Merge remote-tracking branch 'upstream/master' commit dd3b8212ba6fa8b88ccec876b488942abfec437e Author: Aliaksei Sandryhaila <aliaksei.sandryhaila@hp.com> Date: 2015-09-18T21:38:11Z Work in progress: added [incomplete] functionality for memory estimation. commit 0277eb695c8f61a72ec192c8fc57d17e02d1c34b Author: Aliaksei Sandryhaila <aliaksei.sandryhaila@hp.com> Date: 2015-09-18T21:41:53Z Work in progress: adding memory estimation functionality. commit d051d0e73ccee4ee2113f8de625b895117937db6 Author: Aliaksei Sandryhaila <aliaksei.sandryhaila@hp.com> Date: 2015-09-18T21:49:45Z Merge remote-tracking branch 'upstream/master' commit 822a9da76370b97e6231693c6539d677175b3193 Author: Aliaksei Sandryhaila <aliaksei.sandryhaila@hp.com> Date: 2015-09-23T15:46:37Z [Work in progress] Remove debugging code, modify unit tests. commit 03795cd26c637e0cc5c722463bbdebc11000f09d Author: Aliaksei Sandryhaila <aliaksei.sandryhaila@hp.com> Date: 2015-10-12T13:41:15Z Merge remote-tracking branch 'upstream/master' commit 141b56a837e665c2176fc528b6c619e663872776 Author: Aliaksei Sandryhaila <aliaksei.sandryhaila@hp.com> Date: 2015-10-13T18:47:01Z Fixed ORC-21: Added functionality to estimate dynamic memory requirements. commit f5f7ef340e3b20db6d3385be186da54ac49beb41 Author: Aliaksei Sandryhaila <aliaksei.sandryhaila@hp.com> Date: 2015-10-13T18:50:45Z Removed debug code. commit 2c92d49990e76b6a97aa4be7f0400122333e0810 Author: Aliaksei Sandryhaila <aliaksei.sandryhaila@hp.com> Date: 2015-10-13T19:38:26Z Minor code clean-up. commit 219b93a42694870547fcc7f5f5682dd5efb4d59e Author: Aliaksei Sandryhaila <aliaksei.sandryhaila@hp.com> Date: 2015-10-13T19:47:49Z More clean-up. ----
3. Github user omalley commented on the pull request: https://github.com/apache/orc/pull/10#issuecomment-154446961 Sorry, it has taken me so long to get to this patch. You've been very patient. I tried to rebase & squash this down to a single commit and it has conflicts. Can you squash this down to a single commit that applies cleanly to apache/master?
4. GitHub user asandryh opened a pull request: https://github.com/apache/orc/pull/12 Fixed ORC-21: Add dynamic memory usage estimation. An upper bound on memory requirements is provided by two components: * Reader::memoryUse() returns an upper bound on its memory needs. It depends on the file and columns read. * ColumnBatch::memoryUse() returns an upper bound on its memory needs. It

depends on the file, columns, and number of rows read. The new utility FileMemory.cc compares estimated and actual memory usage. You can merge this pull request into a Git repository by running: $ git pull https://github.com/asandryh/orc orc-21 Alternatively you can review and apply these changes as the patch at: https://github.com/apache/orc/pull/12.patch To close this pull request, make a commit to your master/trunk branch with (at least) the following in the commit message: This closes #12 ---- commit 01ea2074b413d7f95664338dc6df7d20224370f3 Author: Aliaksei Sandryhaila <aliaksei.sandryhaila@hp.com> Date: 2015-11-06T19:59:08Z Fixed ORC-21: Add dynamic memory usage estimation. ----

5. Github user asandryh closed the pull request at: https://github.com/apache/orc/pull/10
6. Github user omalley commented on a diff in the pull request: https://github.com/apache/orc/pull/12#discussion_r44199528 --- Diff: c++/include/orc/Vector.hh --- @@ -140,6 +140,11 @@ namespace orc { */ virtual void resize(uint64_t capacity); + /** + * Heap memory used by the batch. + */ + virtual int64_t memoryUse(); --- End diff -- I think the use of -1 as a special value is problematic. I think you should have it return the current usage for all types and add a method hasVariableLength() that returns if it contains a list or map. Please use uint64_t. Please rename to getMemoryUsage().
7. Github user omalley commented on a diff in the pull request: https://github.com/apache/orc/pull/12#discussion_r44199545 --- Diff: c++/include/orc/Reader.hh --- @@ -709,6 +709,16 @@ namespace orc { * @return a string of bytes with the file tail */ virtual std::string getSerializedFileTail() const = 0; + + /** + * Estimate an upper bound on heap memory allocation by the Reader + * based on the information in the file footer. + * The bound is less tight if only few columns are read or compression is used. + * @param stripeIx index of the stripe to be read (if not specified, + * all stripes are considered). + * @return upper bound on memory use + */ + virtual uint64_t memoryUse(int stripeIx=-1) = 0; --- End diff -- Please use getMemoryUse().
8. Github user omalley commented on a diff in the pull request: https://github.com/apache/orc/pull/12#discussion_r44199578 --- Diff: c++/src/Reader.cc --- @@ -1061,7 +1063,6 @@ namespace orc { schema = convertType(footer->types(0), *footer); schema->assignIds(0); - previousRow = (std::numeric_limits<uint64_t>::max)(); --- End diff -- This is a bad change.
9. Github user omalley commented on a diff in the pull request: https://github.com/apache/orc/pull/12#discussion_r44199639 --- Diff: c++/src/Reader.cc --- @@ -1081,8 +1081,8 @@ namespace orc { } void ReaderImpl::selectType(const Type& type) { - if (!selectedColumns[static_cast<size_t>(type.getColumnId())]) { - selectedColumns[static_cast<size_t>(type.getColumnId())] = true; + if (!selectedColumns[type.getColumnId()]) { --- End diff -- Removing these casts causes clang breakage.
10. Github user omalley commented on a diff in the pull request: https://github.com/apache/orc/pull/12#discussion_r44199663 --- Diff: c++/src/Reader.cc --- @@ -1070,9 +1071,8 @@ namespace orc { columnId != included.end(); ++columnId) { if (*columnId == 0) { selectType(*(schema.get())); - } else if (*columnId <= - static_cast<int64_t>(schema->getSubtypeCount())) { - selectType(schema->getSubtype(static_cast<uint64_t>(*columnId-1))); + } else if (*columnId <= static_cast<int64_t>(schema->getSubtypeCount())) { + selectType(schema->getSubtype(*columnId-1)); --- End diff -- This is also wrong.
11. Github user omalley commented on a diff in the pull request: https://github.com/apache/orc/pull/12#discussion_r44199766 --- Diff: c++/src/Reader.cc --- @@ -1364,6 +1364,111 @@ namespace orc { int64_t getEpochOffset() const override; }; + uint64_t maxStreamsForType(const proto::Type& type) { + switch (type.kind()) { + case proto::Type_Kind_STRUCT: + return 1; + case proto::Type_Kind_INT: + case proto::Type_Kind_LONG: + case proto::Type_Kind_SHORT: + case proto::Type_Kind_FLOAT: + case proto::Type_Kind_DOUBLE: + case proto::Type_Kind_BOOLEAN: + case proto::Type_Kind_BYTE: + case proto::Type_Kind_DATE: + case proto::Type_Kind_LIST: + case proto::Type_Kind_MAP: + case proto::Type_Kind_UNION: + return 2; + case proto::Type_Kind_BINARY: + case proto::Type_Kind_DECIMAL: + case proto::Type_Kind_TIMESTAMP: + return 3; + case proto::Type_Kind_CHAR: + case proto::Type_Kind_STRING: + case proto::Type_Kind_VARCHAR: + return 4; + default: + return 0; + } + } + + uint64_t ReaderImpl::memoryUse(int stripeIx) { + uint64_t maxDataLength = 0; + + if (stripeIx >= 0 && stripeIx < footer->stripes_size()) { + uint64_t stripe = footer->stripes(stripeIx).datalength(); + if (maxDataLength < stripe) { + maxDataLength = stripe; + } + } else { + for (int i=0; i < footer->stripes_size(); i++) { + uint64_t stripe = footer->stripes(i).datalength(); + if (maxDataLength < stripe) { + maxDataLength = stripe; + } + } + } + + bool hasStringColumn = false; + uint64_t nSelectedStreams = 0; + for (int i=0; !hasStringColumn && i < footer->types_size(); i++) { + if (selectedColumns[i]) { + const proto::Type& type = footer->types(i); + nSelectedStreams +=

maxStreamsForType(type) ; + switch (type.kind()) { --- End diff -- you need to static cast the kind to int in order to avoid a clang warning.

12. Github user omalley commented on a diff in the pull request: https://github.com/apache/orc/pull/12#discussion_r44199813 --- Diff: c++/src/Reader.cc --- @@ -1433,6 +1538,7 @@ namespace orc { } void ReaderImpl::startNextStripe() { + reader.reset(); // ColumnReaders use lots of memory; free old memory first --- End diff -- I'm not sure, but this likely causes a performance regression.
13. Github user asandryh commented on a diff in the pull request: https://github.com/apache/orc/pull/12#discussion_r44280592 --- Diff: c++/include/orc/Vector.hh --- @@ -140,6 +140,11 @@ namespace orc { */ virtual void resize(uint64_t capacity); + /** + * Heap memory used by the batch. + */ + virtual int64_t memoryUse(); --- End diff -- Sure, using a separate method is cleaner than returning -1. Just to confirm: getMemoryUsage() or getMemoryUse()?
14. Github user asandryh commented on a diff in the pull request: https://github.com/apache/orc/pull/12#discussion_r44281983 --- Diff: c++/src/Reader.cc --- @@ -1061,7 +1063,6 @@ namespace orc { schema = convertType(footer->types(0), *footer); schema->assignIds(0); - previousRow = (std::numeric_limits<uint64_t>::max)(); --- End diff -- Why? In the code immediately above (around lines 1055-1060) we already set previousRow. This line is a left-over from a commit I made months ago when I implemented ReaderImpl::seekToRow() method. I forgot to remove it. I discovered it recently while testing because it conflicts with the logic of startNextStripe() and seekToRow() methods.
15. Github user asandryh commented on a diff in the pull request: https://github.com/apache/orc/pull/12#discussion_r44282151 --- Diff: c++/src/Reader.cc --- @@ -1070,9 +1071,8 @@ namespace orc { columnId != included.end(); ++columnId) { if (*columnId == 0) { selectType(*(schema.get())); - } else if (*columnId <= - static_cast<int64_t>(schema->getSubtypeCount())) { - selectType(schema->getSubtype(static_cast<uint64_t>(*columnId-1))); + } else if (*columnId <= static_cast<int64_t>(schema->getSubtypeCount())) { + selectType(schema->getSubtype(*columnId-1)); --- End diff -- Sorry, that's my mistake. Will restore the cast.
16. Github user asandryh commented on a diff in the pull request: https://github.com/apache/orc/pull/12#discussion_r44282193 --- Diff: c++/src/Reader.cc --- @@ -1081,8 +1081,8 @@ namespace orc { } void ReaderImpl::selectType(const Type& type) { - if (!selectedColumns[static_cast<size_t>(type.getColumnId())]) { - selectedColumns[static_cast<size_t>(type.getColumnId())] = true; + if (!selectedColumns[type.getColumnId()]) { --- End diff -- Will fix this, too.
17. Github user asandryh commented on a diff in the pull request: https://github.com/apache/orc/pull/12#discussion_r44282290 --- Diff: c++/src/Reader.cc --- @@ -1364,6 +1364,111 @@ namespace orc { int64_t getEpochOffset() const override; }; + uint64_t maxStreamsForType(const proto::Type& type) { + switch (type.kind()) { + case proto::Type_Kind_STRUCT: + return 1; + case proto::Type_Kind_INT: + case proto::Type_Kind_LONG: + case proto::Type_Kind_SHORT: + case proto::Type_Kind_FLOAT: + case proto::Type_Kind_DOUBLE: + case proto::Type_Kind_BOOLEAN: + case proto::Type_Kind_BYTE: + case proto::Type_Kind_DATE: + case proto::Type_Kind_LIST: + case proto::Type_Kind_MAP: + case proto::Type_Kind_UNION: + return 2; + case proto::Type_Kind_BINARY: + case proto::Type_Kind_DECIMAL: + case proto::Type_Kind_TIMESTAMP: + return 3; + case proto::Type_Kind_CHAR: + case proto::Type_Kind_STRING: + case proto::Type_Kind_VARCHAR: + return 4; + default: + return 0; + } + } + + uint64_t ReaderImpl::memoryUse(int stripeIx) { + uint64_t maxDataLength = 0; + + if (stripeIx >= 0 && stripeIx < footer->stripes_size()) { + uint64_t stripe = footer->stripes(stripeIx).datalength(); + if (maxDataLength < stripe) { + maxDataLength = stripe; + } + } else { + for (int i=0; i < footer->stripes_size(); i++) { + uint64_t stripe = footer->stripes(i).datalength(); + if (maxDataLength < stripe) { + maxDataLength = stripe; + } + } + } + + bool hasStringColumn = false; + uint64_t nSelectedStreams = 0; + for (int i=0; !hasStringColumn && i < footer->types_size(); i++) { + if (selectedColumns[i]) { + const proto::Type& type = footer->types(i); + nSelectedStreams += maxStreamsForType(type) ; + switch (type.kind()) { --- End diff -- Yes, will add casting here.
18. Github user asandryh commented on a diff in the pull request: https://github.com/apache/orc/pull/12#discussion_r44282877 --- Diff: c++/src/Reader.cc --- @@ -1433,6 +1538,7 @@ namespace orc { } void ReaderImpl::startNextStripe() { + reader.reset(); // ColumnReaders use lots of memory; free old memory first --- End diff -- Why? On line 1550, we assign reader a new value. It destroys the previous value anyway. What calling reset() explicitly does for us is it frees up all the dynamic memory used in the existing instance of ColumnReader BEFORE creating a new ColumnReader and allocating all the buffers. Without calling reset(), the deallocation happens AFTER a

new reader is created, so for a brief moment we use 2x memory than we really need. It messes up our memory estimates, too.

19. Github user omalley commented on a diff in the pull request: https://github.com/apache/orc/pull/12#discussion_r44303166 --- Diff: c++/include/orc/Vector.hh --- @@ -140,6 +140,11 @@ namespace orc { */ virtual void resize(uint64_t capacity); + /** + * Heap memory used by the batch. + */ + virtual int64_t memoryUse(); --- End diff -- I'd prefer getMemoryUsage, but either is ok.
20. Github user omalley commented on a diff in the pull request: https://github.com/apache/orc/pull/12#discussion_r44303551 --- Diff: c++/src/Reader.cc --- @@ -1061,7 +1063,6 @@ namespace orc { schema = convertType(footer->types(0), *footer); schema->assignIds(0); - previousRow = (std::numeric_limits<uint64_t>::max)(); --- End diff -- sorry, I missed that.
21. Github user omalley commented on a diff in the pull request: https://github.com/apache/orc/pull/12#discussion_r44304442 --- Diff: c++/src/Reader.cc --- @@ -1433,6 +1538,7 @@ namespace orc { } void ReaderImpl::startNextStripe() { + reader.reset(); // ColumnReaders use lots of memory; free old memory first --- End diff -- Sorry, I missed that it was the reset on the unique_ptr rather than the ColumnReader. In my profiling, one of the major costs for files with little stripe sizes is rebuilding the ColumnReader and the associated buffer. At some point, we need to clean it up so that it reuses the current ColumnReader and buffers.
22. Github user omalley commented on a diff in the pull request: https://github.com/apache/orc/pull/12#discussion_r44307026 --- Diff: c++/src/Reader.cc --- @@ -1364,6 +1365,111 @@ namespace orc { int64_t getEpochOffset() const override; }; + uint64_t maxStreamsForType(const proto::Type& type) { + switch (static_cast<int64_t>(type.kind())) { + case proto::Type_Kind_STRUCT: + return 1; + case proto::Type_Kind_INT: + case proto::Type_Kind_LONG: + case proto::Type_Kind_SHORT: + case proto::Type_Kind_FLOAT: + case proto::Type_Kind_DOUBLE: + case proto::Type_Kind_BOOLEAN: + case proto::Type_Kind_BYTE: + case proto::Type_Kind_DATE: + case proto::Type_Kind_LIST: + case proto::Type_Kind_MAP: + case proto::Type_Kind_UNION: + return 2; + case proto::Type_Kind_BINARY: + case proto::Type_Kind_DECIMAL: + case proto::Type_Kind_TIMESTAMP: + return 3; + case proto::Type_Kind_CHAR: + case proto::Type_Kind_STRING: + case proto::Type_Kind_VARCHAR: + return 4; + default: + return 0; + } + } + + uint64_t ReaderImpl::getMemoryUse(int stripeIx) { + uint64_t maxDataLength = 0; + + if (stripeIx >= 0 && stripeIx < footer->stripes_size()) { + uint64_t stripe = footer->stripes(stripeIx).datalength(); + if (maxDataLength < stripe) { + maxDataLength = stripe; + } + } else { + for (int i=0; i < footer->stripes_size(); i++) { + uint64_t stripe = footer->stripes(i).datalength(); + if (maxDataLength < stripe) { + maxDataLength = stripe; + } + } + } + + bool hasStringColumn = false; + uint64_t nSelectedStreams = 0; + for (int i=0; !hasStringColumn && i < footer->types_size(); i++) { + if (selectedColumns[i]) { --- End diff -- clang needs: if (selectedColumns[static_cast<size_t>(i)]) { to avoid a message about using signed int where an unsigned int is needed.
23. Github user omalley commented on a diff in the pull request: https://github.com/apache/orc/pull/12#discussion_r44307077 --- Diff: c++/src/Reader.cc --- @@ -1364,6 +1365,111 @@ namespace orc { int64_t getEpochOffset() const override; }; + uint64_t maxStreamsForType(const proto::Type& type) { + switch (static_cast<int64_t>(type.kind())) { + case proto::Type_Kind_STRUCT: + return 1; + case proto::Type_Kind_INT: + case proto::Type_Kind_LONG: + case proto::Type_Kind_SHORT: + case proto::Type_Kind_FLOAT: + case proto::Type_Kind_DOUBLE: + case proto::Type_Kind_BOOLEAN: + case proto::Type_Kind_BYTE: + case proto::Type_Kind_DATE: + case proto::Type_Kind_LIST: + case proto::Type_Kind_MAP: + case proto::Type_Kind_UNION: + return 2; + case proto::Type_Kind_BINARY: + case proto::Type_Kind_DECIMAL: + case proto::Type_Kind_TIMESTAMP: + return 3; + case proto::Type_Kind_CHAR: + case proto::Type_Kind_STRING: + case proto::Type_Kind_VARCHAR: + return 4; + default: + return 0; + } + } + + uint64_t ReaderImpl::getMemoryUse(int stripeIx) { + uint64_t maxDataLength = 0; + + if (stripeIx >= 0 && stripeIx < footer->stripes_size()) { + uint64_t stripe = footer->stripes(stripeIx).datalength(); + if (maxDataLength < stripe) { + maxDataLength = stripe; + } + } else { + for (int i=0; i < footer->stripes_size(); i++) { + uint64_t stripe = footer->stripes(i).datalength(); + if (maxDataLength < stripe) { + maxDataLength = stripe; + } + } + } + + bool hasStringColumn = false; + uint64_t nSelectedStreams = 0; + for (int i=0; !hasStringColumn && i < footer->types_size(); i++) { + if (selectedColumns[i]) { + const proto::Type& type = footer->types(i); + nSelectedStreams += maxStreamsForType(type) ; + switch (static_cast<int64_t>(type.kind())) { + case proto::Type_Kind_CHAR: + case proto::Type_Kind_STRING: + case proto::Type_Kind_VARCHAR: + case proto::Type_Kind_BINARY: { + hasStringColumn = true; + break; + } + default: { + break; + } + } + } + } + + /* If a string column is read, use stripe datalength as a memory estimate + * because we don't

know the dictionary size. Multiply by 2 because + * a string column requires two buffers: + * in the input stream and in the seekable input stream. + * If no string column is read, estimate from the number of streams. + */ + uint64_t memory = hasStringColumn ? 2 * maxDataLength : + std::min(uint64_t(maxDataLength), + nSelectedStreams * stream->getNaturalReadSize()); + + // Do we need even more memory to read the footer or the metadata? + if (memory < postscript->footerlength() + DIRECTORY_SIZE_GUESS) { + memory = postscript->footerlength() + DIRECTORY_SIZE_GUESS; + } + if (memory < postscript->metadatalength()) { + memory = postscript->metadatalength(); + } + + // Account for firstRowOfStripe. + memory += firstRowOfStripe.capacity() * sizeof(uint64_t); + + // Decompressors need buffers for each stream + uint64_t decompressorMemory = 0; + if (compression != CompressionKind_NONE) { + for (int i=0; i < footer->types_size(); i++) { + if (selectedColumns[i]) { --- End diff -- you need the static cast to size_t here too.

24. Github user omalley commented on a diff in the pull request: https://github.com/apache/orc/pull/12#discussion_r44307164 --- Diff: c++/src/Vector.cc --- @@ -196,6 +246,17 @@ namespace orc { } } + uint64_t MapVectorBatch::getMemoryUsage() { + return ColumnVectorBatch::getMemoryUsage() + + static_cast<uint64_t>(offsets.capacity() * sizeof(int64_t)) + + keys->getMemoryUsage(); --- End diff -- You have a spurious ';' here that is making your results and tests wrong.
25. Github user omalley commented on the pull request: https://github.com/apache/orc/pull/12#issuecomment-155148205 I pushed my fixes for this patch to omalley/orc-21.
26. Github user asandryh commented on a diff in the pull request: https://github.com/apache/orc/pull/12#discussion_r44314198 --- Diff: c++/src/Vector.cc --- @@ -196,6 +246,17 @@ namespace orc { } } + uint64_t MapVectorBatch::getMemoryUsage() { + return ColumnVectorBatch::getMemoryUsage() + + static_cast<uint64_t>(offsets.capacity() * sizeof(int64_t)) + + keys->getMemoryUsage(); --- End diff -- Argggh, this is a silly one. Thank you for catching this bug! I didn't even get a compiler warning on this one.
27. Github user asfgit closed the pull request at: https://github.com/apache/orc/pull/12