Item 340
**git_comments:**

1. log.debug("Calling continue scan : "+scanState.range+" loc = "+loc);

**git_commits:**

1. **summary:** ACCUMULO-4145 Eliminate Text wrapping of tableIDs
   **message:** ACCUMULO-4145 Eliminate Text wrapping of tableIDs
   **label:** code-design

**github_issues:**

**github_issues_comments:**

**github_pulls:**

**github_pulls_comments:**

**github_pulls_reviews:**

**jira_issues:**

1. **summary:** Eliminate constant unnecessary Text wrapping of immutable tableIDs
   **description:** I started looking at ACCUMULO-4138 to see how KeyExtent is used to do overlaps, and I noticed a *lot* of {{new KeyExtent(new Text(String), ...)}} calls, where that first parameter is the Text version of tableId. It appears, we even do this practice so much, that KeyExtent has a built-in WeakReference caching of these tableIds, so the Java GC we can dedupe them and avoid creating too many. The thing is... a lot of this attempt to optimize appears to be the result of unnecessarily wrapping all these immutable String tableIDs with a Text object, yet it doesn't really seem to buy us anything. In most of our API and a lot of internal utilities, these are already Strings with references elsewhere in the code... so even if we dedupe the Text objects, we're not doing anything about these Strings. Worse, we're actually doing some protective copying here and there as we pass around these Text objects, using TextUtil, etc. This is completely unnecessary, because they were immutable before we copied them and wrapped them. In some cases, we actually call toString on the text objects to pass them around, and they get flip-flopped a few times between String and Text, depending on what the internal API accepts. At best, using the Text object helps us serialize KeyExtent... but that's questionably beneficial since DataOutput can already serialize with: {{out.writeUTF(String)}}, so that's not actually that helpful. The only other benefit I could possibly see is with compareTo for lexicographical comparison, but I have a hard time believing Text can compare faster than {{java.lang.String}}, and there shouldn't be any difference in the results of the comparison between the UTF-8 encoding of Text and the modified UTF encoding which is native to Java Strings, certainly not for the base-36 characters and fixed constants (for special tables) we use in tableIDs. We should just completely strip out all the Text versions of tableId, wherever possible. I've already done this as an exercise in the 1.6 branch, but it might be potentially risky as these are put in Java maps which don't have strict type checking ({{map.get(Object)}}, for instance) and are sometimes compared with {{.equals(Object)}}. For what it's worth, the only public API this really touches is {{o.a.a.core.data.KeyExtent}}, which was only inadvertently in the public API and has since been moved (IIRC). We can preserve the old methods for compatibility, if necessary (deprecated, of course). Also, getting rid of these Text objects, and just sticking with the immutable String objects, we'll be able to take advantage of future JVM improvements for String deduplication, like http://java-performance.info/java-string-deduplication/
2. **summary:** Eliminate constant unnecessary Text wrapping of immutable tableIDs
   **description:** I started looking at ACCUMULO-4138 to see how KeyExtent is used to do overlaps, and I noticed a *lot* of {{new KeyExtent(new Text(String), ...)}} calls, where that first parameter is the Text version of tableId. It appears, we even do this practice so much, that KeyExtent has a built-in WeakReference caching of these tableIds, so the Java GC we can dedupe them and avoid creating too many. The thing is... a lot of this attempt to optimize appears to be the result of unnecessarily wrapping all these immutable String tableIDs with a Text object, yet it doesn't really seem to buy us anything. In most of our API and a lot of internal utilities, these are already Strings with references elsewhere in the

code... so even if we dedupe the Text objects, we're not doing anything about these Strings. Worse, we're actually doing some protective copying here and there as we pass around these Text objects, using TextUtil, etc. This is completely unnecessary, because they were immutable before we copied them and wrapped them. In some cases, we actually call toString on the text objects to pass them around, and they get flip-flopped a few times between String and Text, depending on what the internal API accepts. At best, using the Text object helps us serialize KeyExtent... but that's questionably beneficial since DataOutput can already serialize with: {{out.writeUTF(String)}}, so that's not actually that helpful. The only other benefit I could possibly see is with compareTo for lexicographical comparison, but I have a hard time believing Text can compare faster than {{java.lang.String}}, and there shouldn't be any difference in the results of the comparison between the UTF-8 encoding of Text and the modified UTF encoding which is native to Java Strings, certainly not for the base-36 characters and fixed constants (for special tables) we use in tableIDs. We should just completely strip out all the Text versions of tableId, wherever possible. I've already done this as an exercise in the 1.6 branch, but it might be potentially risky as these are put in Java maps which don't have strict type checking ({{map.get(Object)}}, for instance) and are sometimes compared with {{.equals(Object)}}. For what it's worth, the only public API this really touches is {{o.a.a.core.data.KeyExtent}}, which was only inadvertently in the public API and has since been moved (IIRC). We can preserve the old methods for compatibility, if necessary (deprecated, of course). Also, getting rid of these Text objects, and just sticking with the immutable String objects, we'll be able to take advantage of future JVM improvements for String deduplication, like http://java-performance.info/java-string-deduplication/
**label:** code-design

3. **summary:** Eliminate constant unnecessary Text wrapping of immutable tableIDs
**description:** I started looking at ACCUMULO-4138 to see how KeyExtent is used to do overlaps, and I noticed a *lot* of {{new KeyExtent(new Text(String), ...)}} calls, where that first parameter is the Text version of tableId. It appears, we even do this practice so much, that KeyExtent has a built-in WeakReference caching of these tableIds, so the Java GC we can dedupe them and avoid creating too many. The thing is... a lot of this attempt to optimize appears to be the result of unnecessarily wrapping all these immutable String tableIDs with a Text object, yet it doesn't really seem to buy us anything. In most of our API and a lot of internal utilities, these are already Strings with references elsewhere in the code... so even if we dedupe the Text objects, we're not doing anything about these Strings. Worse, we're actually doing some protective copying here and there as we pass around these Text objects, using TextUtil, etc. This is completely unnecessary, because they were immutable before we copied them and wrapped them. In some cases, we actually call toString on the text objects to pass them around, and they get flip-flopped a few times between String and Text, depending on what the internal API accepts. At best, using the Text object helps us serialize KeyExtent... but that's questionably beneficial since DataOutput can already serialize with: {{out.writeUTF(String)}}, so that's not actually that helpful. The only other benefit I could possibly see is with compareTo for lexicographical comparison, but I have a hard time believing Text can compare faster than {{java.lang.String}}, and there shouldn't be any difference in the results of the comparison between the UTF-8 encoding of Text and the modified UTF encoding which is native to Java Strings, certainly not for the base-36 characters and fixed constants (for special tables) we use in tableIDs. We should just completely strip out all the Text versions of tableId, wherever possible. I've already done this as an exercise in the 1.6 branch, but it might be potentially risky as these are put in Java maps which don't have strict type checking ({{map.get(Object)}}, for instance) and are sometimes compared with {{.equals(Object)}}. For what it's worth, the only public API this really touches is {{o.a.a.core.data.KeyExtent}}, which was only inadvertently in the public API and has since been moved (IIRC). We can preserve the old methods for compatibility, if necessary (deprecated, of course). Also, getting rid of these Text objects, and just sticking with the immutable String objects, we'll be able to take advantage of future JVM improvements for String deduplication, like http://java-performance.info/java-string-deduplication/
4. **summary:** Eliminate constant unnecessary Text wrapping of immutable tableIDs
**description:** I started looking at ACCUMULO-4138 to see how KeyExtent is used to do overlaps, and I noticed a *lot* of {{new KeyExtent(new Text(String), ...)}} calls, where that first parameter is the Text version of tableId. It appears, we even do this practice so much, that KeyExtent has a built-in WeakReference caching of these tableIds, so the Java GC we can dedupe them and avoid creating too many. The thing is... a lot of this attempt to optimize appears to be the result of unnecessarily wrapping all these immutable String tableIDs with a Text object, yet it doesn't really seem to buy us anything. In most of our API and a lot of internal utilities, these are already Strings with references elsewhere in the code... so even if we dedupe the Text objects, we're not doing anything about these Strings. Worse, we're actually doing some protective copying here and there as we pass around these Text objects, using

TextUtil, etc. This is completely unnecessary, because they were immutable before we copied them and wrapped them. In some cases, we actually call toString on the text objects to pass them around, and they get flip-flopped a few times between String and Text, depending on what the internal API accepts. At best, using the Text object helps us serialize KeyExtent... but that's questionably beneficial since DataOutput can already serialize with: {{out.writeUTF(String)}}, so that's not actually that helpful. The only other benefit I could possibly see is with compareTo for lexicographical comparison, but I have a hard time believing Text can compare faster than {{java.lang.String}}, and there shouldn't be any difference in the results of the comparison between the UTF-8 encoding of Text and the modified UTF encoding which is native to Java Strings, certainly not for the base-36 characters and fixed constants (for special tables) we use in tableIDs. We should just completely strip out all the Text versions of tableId, wherever possible. I've already done this as an exercise in the 1.6 branch, but it might be potentially risky as these are put in Java maps which don't have strict type checking ({{map.get(Object)}}, for instance) and are sometimes compared with {{.equals(Object)}}. For what it's worth, the only public API this really touches is {{o.a.a.core.data.KeyExtent}}, which was only inadvertently in the public API and has since been moved (IIRC). We can preserve the old methods for compatibility, if necessary (deprecated, of course). Also, getting rid of these Text objects, and just sticking with the immutable String objects, we'll be able to take advantage of future JVM improvements for String deduplication, like http://java-performance.info/java-string-deduplication/
**label:** code-design

5. **summary:** Eliminate constant unnecessary Text wrapping of immutable tableIDs
**description:** I started looking at ACCUMULO-4138 to see how KeyExtent is used to do overlaps, and I noticed a *lot* of {{new KeyExtent(new Text(String), ...)}} calls, where that first parameter is the Text version of tableId. It appears, we even do this practice so much, that KeyExtent has a built-in WeakReference caching of these tableIds, so the Java GC we can dedupe them and avoid creating too many. The thing is... a lot of this attempt to optimize appears to be the result of unnecessarily wrapping all these immutable String tableIDs with a Text object, yet it doesn't really seem to buy us anything. In most of our API and a lot of internal utilities, these are already Strings with references elsewhere in the code... so even if we dedupe the Text objects, we're not doing anything about these Strings. Worse, we're actually doing some protective copying here and there as we pass around these Text objects, using TextUtil, etc. This is completely unnecessary, because they were immutable before we copied them and wrapped them. In some cases, we actually call toString on the text objects to pass them around, and they get flip-flopped a few times between String and Text, depending on what the internal API accepts. At best, using the Text object helps us serialize KeyExtent... but that's questionably beneficial since DataOutput can already serialize with: {{out.writeUTF(String)}}, so that's not actually that helpful. The only other benefit I could possibly see is with compareTo for lexicographical comparison, but I have a hard time believing Text can compare faster than {{java.lang.String}}, and there shouldn't be any difference in the results of the comparison between the UTF-8 encoding of Text and the modified UTF encoding which is native to Java Strings, certainly not for the base-36 characters and fixed constants (for special tables) we use in tableIDs. We should just completely strip out all the Text versions of tableId, wherever possible. I've already done this as an exercise in the 1.6 branch, but it might be potentially risky as these are put in Java maps which don't have strict type checking ({{map.get(Object)}}, for instance) and are sometimes compared with {{.equals(Object)}}. For what it's worth, the only public API this really touches is {{o.a.a.core.data.KeyExtent}}, which was only inadvertently in the public API and has since been moved (IIRC). We can preserve the old methods for compatibility, if necessary (deprecated, of course). Also, getting rid of these Text objects, and just sticking with the immutable String objects, we'll be able to take advantage of future JVM improvements for String deduplication, like http://java-performance.info/java-string-deduplication/
**label:** code-design

**jira_issues_comments:**

1. GitHub user ctubbsii opened a pull request: https://github.com/apache/accumulo/pull/70 ACCUMULO-4145 Eliminate Text wrapping of tableIDs Best effort attempt to eliminate Text wrapping of tableIDs for 1.6 branch, to example the extent of the changes. Preserves KeyExtent caching with WeakHashMap (might not be needed, or possibly could be replaced with String.intern()). Preserves Text serialization/deserialization of tableIDs in KeyExtent serialization/deserialization (needed for backwards compatibility and KeyExtents serialized in WALs). You can merge this pull request into a Git repository by running: $ git pull https://github.com/ctubbsii/accumulo ACCUMULO-4145-string-tableIDs Alternatively you can review and apply these changes as the patch at:

https://github.com/apache/accumulo/pull/70.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 #70 ---- commit fbad09c7cd42709cebbcd35e23ddbd58fb45aea5 Author: Christopher Tubbs <ctubbsii@apache.org> Date: 2016-02-15T06:56:45Z ACCUMULO-4145 Eliminate Text wrapping of tableIDs ----

2. Github user ctubbsii commented on the pull request: https://github.com/apache/accumulo/pull/70#issuecomment-184089574 For reference, String.intern() should not be used for Java 1.6 runtime, as that could grow the PermGen space significantly. It could possibly replace the WeakHashMap in 1.7, because intern storage has been moved to the main Java heap. It might not actually be needed at all, since we've effectively eliminated all the extra objects and copies that were left around with Text. And, in Java 1.8 update 30, the G1 GC has a String dedupe feature which may eliminate the extra copies without needing to intern.
3. I double checked. KeyExtent is not in the public API (confirmed with README), so this would change no public API... at least not in 1.6. I'd have to double check to see if it affects any public API in newer branches. We first need to determine if we'd want to go ahead and do this in 1.6/1.7 or wait until 1.8.
4. **body:** KeyExtent is referenced by methods that are in the public API. During the 1.7.0 API cleanup I did the following for KeyExtent. * Deprecated data.KeyExtent and all public API methods that use it. * Created TabletId class in public API and replaced deprecated methods that used KeyExtent with new ones that used TabletId * Created a new data.impl.KeyExtent class that all internal code uses. Only deprecated APIs use data.KeyExtent now * Made data.KeyExtent wrap data.impl.KeyExtent I made data.KeyExtent wrap data.impl.KeyExtent instead of extend so that when new methods are added to data.impl.KeyExtent, data.KeyExtent does not automatically inherit those methods. This freezes the deprecated KeyExtent API. Given that in 1.7.0 I tried to be really conservative with API changes and minimize the possibility of breaking user code, I would not be in favor of these changes for 1.6. The GeoWave input format uses tablet locator and key extent (until 1.8.0 is realesed with tablet location in the API).
   **label:** code-design
5. Okay, so the safest thing is to wait until 1.8, and to avoid modifying the frozen, deprecated KeyExtent API.
6. Github user ctubbsii closed the pull request at: https://github.com/apache/accumulo/pull/70
7. GitHub user ctubbsii opened a pull request: https://github.com/apache/accumulo/pull/72 ACCUMULO-4145 Eliminate Text wrapping of tableIDs * Best attempt to eliminate text wrapping of tableIDs on master branch. * Some ByteBuffer wrapping still occurs due to thrift using bytes to transfer tableId instead of string. * Some Text wrapping is needed for: - preserving existing KeyExtent serialization - putting tableIDs in replication-related table entries - avoiding public API changes to deprecated KeyExtent. All unit tests pass, and -Psunny ITs, as well as checkstyle, findbugs, and modernizer This replaces #70 You can merge this pull request into a Git repository by running: $ git pull https://github.com/ctubbsii/accumulo ACCUMULO-4145 Alternatively you can review and apply these changes as the patch at: https://github.com/apache/accumulo/pull/72.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 #72 ---- commit 413244765adb7a60cef8de701de6983d4c99c8e2 Author: Christopher Tubbs <ctubbsii@apache.org> Date: 2016-02-18T23:56:54Z ACCUMULO-4145 Eliminate Text wrapping of tableIDs ----
8. Github user keith-turner commented on a diff in the pull request: https://github.com/apache/accumulo/pull/72#discussion_r53470957 --- Diff: core/src/main/java/org/apache/accumulo/core/client/impl/ScannerImpl.java --- @@ -60,7 +59,7 @@ public ScannerImpl(ClientContext context, String table, Authorizations authoriza checkArgument(table != null, "table is null"); checkArgument(authorizations != null, "authorizations is null"); this.context = context; - this.table = new Text(table); + this.table = table; --- End diff -- seems like this is a tableId... could change the name
9. Github user keith-turner commented on a diff in the pull request: https://github.com/apache/accumulo/pull/72#discussion_r53473276 --- Diff: core/src/main/java/org/apache/accumulo/core/client/impl/TabletLocator.java --- @@ -96,19 +96,19 @@ public static synchronized void clearLocators() { locators.clear(); } - public static synchronized TabletLocator getLocator(ClientContext context, Text tableId) { + public static synchronized TabletLocator getLocator(ClientContext context, String tableId) { --- End diff -- For external projects that are using this, I am happy 1.8 finally has locator in the public API.
10. Github user keith-turner commented on a diff in the pull request: https://github.com/apache/accumulo/pull/72#discussion_r53473836 --- Diff: core/src/test/java/org/apache/accumulo/core/client/impl/TabletLocatorImplTest.java --- @@ -63,10 +63,10 @@ public class TabletLocatorImplTest { private static final KeyExtent RTE =

RootTable.EXTENT; - private static final KeyExtent MTE = new KeyExtent(new Text(MetadataTable.ID), null, RTE.getEndRow()); --- End diff -- nice to see all the `new Text` to go away

11. Github user keith-turner commented on the pull request: https://github.com/apache/accumulo/pull/72#issuecomment-186260830 +1
12. Github user ctubbsii closed the pull request at: https://github.com/apache/accumulo/pull/72