Item 38
git_comments:

git_commits:

1. **summary:** HBASE-669 MultiRegion transactions with Optimistic Concurrency Control; disable test that doesn't pass yet
   message: HBASE-669 MultiRegion transactions with Optimistic Concurrency Control; disable test that doesn't pass yet git-svn-id: https://svn.apache.org/repos/asf/hadoop/hbase/trunk@686663 13f79535-47bb-0310-9956-ffa450edef68

github_issues:

github_issues_comments:

github_pulls:

github_pulls_comments:

github_pulls_reviews:

jira_issues:

1. **summary:** MultiRegion transactions with Optimistic Concurrency Control
   description: We have a need for ACID transactions across tables. This issue is about adding transactions which span multiple regions. We do not envision many competing writes, and will be read-dominated in general. This makes Optimistic Concurrency Control (OCC) seem like the way to go.

jira_issues_comments:

1. This is my current, in-progress, attempt: HRegions keep track of read and write sets for each transaction. To decide if we commit transaction A, we check for overlap between the read set of A and the write set of all transactions that were commited since A started. If there is overlap, then we must abort. If not the we can commit this transaction without conflict (in the given region). Client-side we keep track of all the regions involved in a transaction. When the client asks to commit, we ask all participating regions if it is ok to commit. If so we commit, otherwise we abort. This patch contains a few simple tests to communicate the API. Still missing are Write Ahead Logs (WALs) on clientside and integrating with the HRegion's log. I would appreciate high-level review/feedback on: - clientside API - implications of this OCC approach on the rest of hbase - general implementation approach. Currently I'm subclassing to keep my changes isolated, but this may not be best. EG, I have subclasses of HRegionServer that do both transactions and secondary indexes. How do I get both? A mixin/AOP approach may work better here...
2. **body:** Current version: Added StressTestTransactions which tries to do a lot of conflicting transactions. This was useful in finding some subtle issues with synchronization and the commit protocol, and gives me some confidence that everything is working properly. still TODO: - integrate with regionserver's WAL - come up with global transaction log solution. This log is only read when there has been a failure during the commit process: Regionservers are holding a COMMIT_PENDING transaction (they voted to commit, but have not heard the final, global solution) and never hear back the client. Or a regionserver crashes, restarts, looks at his WAL and sees entries for a transaction, but no final decision (commit or abort). I have a simple interface for the transaction logger client, and I'm considering implementations in hdfs, zookeeper, or a specialized transaction server backed by hdfs. A pure hdfs implementation is attractive because it requires no additional infrastructure/servers, however performance will not be good as we will just be writing lots of small, short lived files. (All we need is ability to get unique transactionId's, and map between transactionId's and transaction state) So now I'm leaning toward a special-purpose server which could run alongside the master...
   label: code-design
3. Couple of quick comments in no particular order after cursory read (so some of the questions might be extra dumb): + The client runs the delete edits to invalidate the transaction? (By replaying the edits out of the client-side WAL)? If so, can it be done regionserver side? If client lease expires or client cancels the transaction, the regionserver does the undo operations? + Good stuff making HTable, etc. subclassable. If

you want to do these changes as a separate patch, I'll just commit it so you don't have to reproduce it in this and your secondary index patch. Make an issue. + On, feedback regards how to have more than one regionserver implementation -- transactional and secondary indices -- I think yeah, probably has to be some mixin factility. I can see this transactional stuff eventually making it into core. + I don't get the global transaction log doohickey. Why not just have regionservers just discard COMMIT_PENDING if lease times out or on restart, comes across a COMMIT_PENDING replaying? + We're trying to get zookeeper into the mix tout-de-suite. Then you wouldn't have to do a custom server? + Classes are missing licenses. Some of those that have them, have 2007 as date. + On '// FIXME, not sure why this is needed, but copied over from HTable', you want to use Arrays.toArray or some such? (I ain't sure why its there either) + Why pass transaction state when doing scan/get reads? + Add a package info under client transactional that explains what this feature is, how it works and how to use it (will help others get invovled). + TransctionalState on server side should inherit from the client TransactionState? They have mostly same enums? + I applied your getBytes.toInt ...HBASE-676 (thanks)

4. I basically followed the original OCC paper with modifications for a distributed commit protocol (because we have multiple parties involved). That paper is available here: http://www.seas.upenn.edu/~zives/cis650/papers/opt-cc.pdf > The client runs the delete edits to invalidate the transaction? No, we don't apply the edits until after the global commit decision has been made. As they come into the regionserver they are WAL'ed and stored with the transaction state until a commit. > I don't get the global transaction log doohickey. Why not just have regionservers just discard COMMIT_PENDING if lease times out or on restart, comes across a COMMIT_PENDING replaying? This is to handle the case where we voted to commit and then either regionserver or the client dies. Its possible that the client told everyone else to commit, so if we disregarded our part of the transaction the we loose the AC from ACID. + We're trying to get zookeeper into the mix tout-de-suite. Then you wouldn't have to do a custom server? Yeah, that would be ideal. > Why pass transaction state when doing scan/get reads? We need this to check for conflicts. If we read a row that was later written (in a transaction that committed in the meantime) the we have to abort the read transaction. > TransctionalState on server side should inherit from the client TransactionState? They have mostly same enums? Yeah this probably makes sense. Thanks for review.
5. > On '// FIXME, not sure why this is needed, but copied over from HTable', you want to use Arrays.toArray or some such? (I ain't sure why its there either) Why can't we just 'return values;'? Seems to me all the following code does is make a copy of the values array: <code> if (values != null) { ArrayList<Cell> cellValues = new ArrayList<Cell>(); for (int i = 0 ; i < values.length; i++) { cellValues.add(values[i]); } return cellValues.toArray(new Cell[values.length]); } return null; </code> maybe thats just remains of a past, no longer needed conversion?
6. I committed change that just returns 'values'. Thanks.
7. **body:** This patch includes an attempt at integrating with the HLog We add log messages from transaction operations START, WRITE, COMMIT, and ABORT. transactional writes are written to the HLog as they are received (before the commit decision has been made). When we actually decide to commit, we just write the COMMIT message to the log, and the apply all of the BatchUpdates *without* logging them. To restore from the log we scan though, keeping track of all the WRITES for a given transactionId, and when we see the COMMIT message we go ahead and apply the writes. If we don't see a COMMIT or ABORT message, the we have to go to the transaction log to see what happened to the transaction. So this means that in order to restore a transaction, we need to look at all of the log entries for the transaction. To ensure this, we need to fiddle with the commitSequenceId that is written as we flush to the HLog. When we flush, we need to use the minimum sequence Id from the START messages from all transactions that are currently pending. That way when we go to recover we will have all those pending transaction's log messages... Adding this behavior was kinda messy because: - HLog recover takes place in each HStore, but I need to recover at the region level (because transactions cross stores). This means the HLog is being scanned once for each HStore which seems ineffecient. I added a hook for the region to do it's log recovery that I use in TransactionalRegion. - HLog/HLogEdit/HLogKey are not easily subclassable. - Log writing concerns are in HLog, while reading is in HStore. - Log recovery took place in the constructor of HRegion. I put this in an initialize() method for the HRegion so that HRegion's subclass constructors can run before they process a log. - startSequence fiddling as described above I have a test to make sure that reading/writing from the log works, but to have any confidence in this I need to test in the broader context where hregion is driving the writes. So I'd like to bring up a cluster, do some operations, be sure that some stuff was not flushed, bring down a regionserver hard, and bring it back up such that it recovers from log. I did not see any such tests of the current WAL.... Would really appreciate a review of this approach from someone more familiar with logging/flushing process than I.
label: code-design

8. Latest patch. Testing of hlog recovery.
9. updated version applies to trunk, fixes a few bugs in prev version
10. newest version. Fixes bug when reading from cells that have been previously written from the same transaction.
11. **body:** I tried to apply last patch Clint. Fails in HTable. I'd like to commit this. It has a bunch of unit tests so its at least basically working. Blocker is lack of package documentation. Please add a package.html (see thrift package for example) or a package-info.java (e.g. mapred package) that explains what this is all about with pointer to the OCC paper and a basic howto on what configuration is needed to make a transactional hbase float. Thanks Clint.
    label: documentation
12. Attaching current version: a few fixes, documentation, and cleanup. I think this is ready for commit. The main remaining issue is correctly recovering from the HLog in event of crash. After this patch is applied, I'll create a new JIRA to track that work. Running full tests now...
13. +1. I have been using this without any issues for the past few weeks.
14. Committed (fixed up a few of the licenses -- said 2007 -- and added on to TestHLogRecover). Thanks for the sweet new feature Clint.
15. > Clint Morgan - 17/Aug/08 12:18 PM > Attaching current version: a few fixes, documentation, and cleanup. > > I think this is ready for commit. The main remaining issue is correctly recovering from the HLog in event > of crash. How would this recovery be different from HBASE-728 combined with HBASE-698 ?
16. When replaying a hlog, we need to track transaction start, write, commit, and abort messages and react appropriately. This contrasts with the normal logic of simply replaying HLog edits at the hstore level. I have this logic in place (see changes to HLog, HLogEdit, HLogKey), but have not fully tested it. There are a couple of tricky/sublte issues there that I'm not sure I've gotten right. EG, the min sequence number that we have to look at for recovery is not simply that of the last flush. Rather it depends on the min seq numbers of the start messages of pending transactions at the time of hlog flush... Also, if we crash at just the right time, we could have committed pieces of a transaction in other regions, but not have the commit message in the log for this region. In this case, we need to lookup the transaction state (commited/aborted) in a global transaction log. I'd like to put this global transaction log in Zookeper when its in the mix.