Item 371
git_comments:

1. * * calls both flush and forceWrite methods if regular flush is enabled. * * @param forceMetadata * - If true then this method is required to force changes to * both the file's content and metadata to be written to storage; * otherwise, it need only force content changes to be written * @throws IOException
2. * Creates a new log file. This method should be guarded by a lock, * so callers of this method should be in right scope of the lock.
3. * in any case that an entry log reaches the limit, we roll the log * and start checkpointing. if a memory table is flushed spanning * over two entry log files, we also roll log. this is for * performance consideration: since we don't wanna checkpoint a new * log file that ledger storage is writing to.
4. * in the case of entrylogperledger, SyncThread drives * checkpoint logic for every flushInterval. So * EntryMemtable doesn't need to call checkpoint in the case * of entrylogperledger.
5. *
6. * add entry to the corresponding entrylog and return the position of * the entry in the entrylog
7. * close current logs.
8. * this method should be called before doing entrymemtable flush, it * would save the state of the entrylogger before entrymemtable flush * and commitEntryMemTableFlush would take appropriate action after * entrymemtable flush.
9. * * flushCurrentLogs method is called during checkpoint, so * metadata of the file also should be force written.
10. log channel can be null because the file is deferred to be created
11. * * Append the ledger map at the end of the entry log. * Updates the entry log file header with the offset and size of the map.
12. * Do the operations required for checkpoint.
13. * this method should be called after doing entrymemtable flush,it would * take appropriate action after entrymemtable flush depending on the * current state of the entrylogger and the state of the entrylogger * during prepareEntryMemTableFlush. * It is assumed that there would be corresponding * prepareEntryMemTableFlush for every commitEntryMemTableFlush and both * would be called from the same thread. * returns boolean value indicating whether EntryMemTable should do checkpoint * after this commit method. boolean commitEntryMemTableFlush() throws IOException; } abstract class EntryLogManagerBase implements EntryLogManager { volatile List<BufferedLogChannel> rotatedLogChannels; private final FastThreadLocal<ByteBuf> sizeBufferForAdd = new FastThreadLocal<ByteBuf>() { @Override protected ByteBuf initialValue() throws Exception { return Unpooled.buffer(4); } }; /* * This method should be guarded by a lock, so callers of this method * should be in the right scope of the lock.
14. lock it only if there is new data so that cache accesstime is not changed
15. If the current entry log disk is full, then create new entry log.
16. * gets the active logChannel with the given entryLogId. null if it is * not existing.
17. * flush current logs.
18. flush the internal buffer back to filesystem but not sync disk
19. * * Datastructure which maintains the status of logchannels. When a * logChannel is created entry of < entryLogId, false > will be made to this * sortedmap and when logChannel is rotated and flushed then the entry is * updated to < entryLogId, true > and all the lowest entries with * < entryLogId, true > status will be removed from the sortedmap. So that way * we could get least unflushed LogId. *
20. * In the case of entryLogPerLedgerEnabled we need to flush * both rotatedlogs and currentlogs. This is needed because * syncThread periodically does checkpoint and at this time * all the logs should be flushed. *
21. log file suffix
22. **comment:** Flush the ledger's map out before we write the header. Otherwise the header might point to something that is not fully written
    label: code-design
23. * flush both current and rotated logs.
24. * flush rotated logs.
25. * prepareSortedLedgerStorageCheckpoint is required for * singleentrylog scenario, but it is not needed for * entrylogperledger scenario, since entries of a ledger go * to a entrylog (even during compaction) and SyncThread * drives periodic checkpoint logic.
26. do nothing
27. * force close current logs.
28. Extracted from createNewLog()
29. Creating a new configuration with a number of ledger directories.
30. wait for the pre-allocation to complete
31. * when entryLogger is created Header of length EntryLogger.LOGFILE_HEADER_SIZE is created
32. * since entrylog/Bufferedchannel is persisted (forcewritten), we should be able to read the entrylog using * newEntryLogger
33. * 'flushIntervalInBytes' number of bytes are flushed so BufferedChannel should be forcewritten
34. entrylogger writes length of the entry (4 bytes) before writing entry
35. since we marked entrylog-5 as rotated, LeastUnflushedLogId would be previous rotatedlog+1
36. entrylog-3 is already rotated, so leastUnflushedLogId should be 4
37. since we marked entrylog - 0 as rotated, LeastUnflushedLogId would be previous rotatedlog+1
38. here though we rotated entrylog-3, entrylog-2 is not yet rotated so LeastUnflushedLogId should be still 2
39. * * Test to verify the leastUnflushedLogId logic in EntryLogsStatus.
40. * test for validating if the EntryLog/BufferedChannel flushes/forcewrite if the bytes written to it are more than * flushIntervalInBytes

git_commits:

1. **summary:** Issue #570: Introducing EntryLogManager.
   message: Issue #570: Introducing EntryLogManager. Descriptions of the changes in this PR: Introducing EntryLogManager interface, which abstracts out current activeLogChannel, rotatedLogChannels and corresponding lock for activeLogChannel. The current logic of handling logs is moved to EntryLogManagerForSingleEntryLog class, in the next sub-task EntryLogManagerForEntryLogPerLedger will be introduced. Also there are minor changes to createNewLog logic and leastUnflushedLogId logic. This is < sub-task5 > of Issue #570 Master Issue: #570 Author: cguttapalem <cguttapalem@salesforce.com> Reviewers: Sijie Guo <sijie@apache.org> This closes #1281 from reddycharan/entrylogmanager, closes #570

github_issues:

1. **title:** Multiple active entrylogs
   body: JIRA: https://issues.apache.org/jira/browse/BOOKKEEPER-1041 Reporter: Venkateswararao Jujjuri (JV) @jvrao Current bookkeeper is tuned for rotational HDDs. It has one active entrylog, and all the ledger/entries go to the same entrylog until it is rotated out. This is

perfect for HDDs as seeks and moving head allover the disk platter is very expensive. But this is very inefficient for SSDs, as each SSD can handle multiple parallel writers, also this method is extremely inefficient for compaction as it causes write amplification and inefficient disk space usage. Our proposal is to have multiple active entrylogs and a configuration param on how many parallel entrylogs the system can have. This way one can have ability to configure to have less (may be one) ledger per entrylog. ### Comments from JIRA --- *Enrico Olivelli* 2017-04-20T07:28:28.619+0000 {quote} But this is very inefficient for HDDs {quote} [~jujjuri] did you mean SSD ? --- *Venkateswararao Jujjuri (JV)* 2017-04-21T16:42:29.203+0000 Yes I mean SSD. Corrected. Thanks. --- *Charan Reddy Guttapalem* 2017-05-18T15:01:22.566+0000 Issue: In Bookie's EntryLogger, we are having only one current active entryLog and all the ledger/entries go to the same entrylog. This is perfect for HDDs as file syncs, seeks and moving head allover the disk platter is very expensive. But having single active Entry Log is inefficient for SSDs, as each SSD can handle multiple parallel writers. Also, having single active EntryLog (irrespective of LedgerStorage type - interleaved/sorted), is inefficient for compaction, since entries of multiple ledgers will end up in an entrylog. Also in SortedLedgerStorage , in the addEntry request we flush EntryMemtable, if it reaches the sizelimit. Because of this we are observing unpredictable tail latency for addEntry request. When EntryMemTable snapshot of size (64 MB) is flushed all at once, this may affect the journal addentry latency. Also, if the rate of new add requests surpasses the rate at which the EntryMemTable's previous snapshot is flushed, then at a point the current EntryMemTable map will reach the limit and since the previous snapshot flush is in progress, EntryMemTable will throttle new addRequests, which would affect addEntry latency. Goals: The main purpose of this feature is to have efficient Garbagecollection story by minimizing the amount of compactions required and the ability to reclaim the deleted ledger's space quicker. Also with this feature we can lay foreground for switching to InterleavedLedgerStorage from SortedLedgerStorage and get predictable tail latency. Proposal: So proposal here is to have multiple active entrylogs. Which will help with compaction performance and make SortedLedgerStorage redundant. Design Overview: - is to have server configuration specifying number of active entry logs per ledgerdir. - for backward compatibility (for existing behaviour) that config can be set to 0. - round-robin method will be used for choosing the active entry log for the current ledger in EntryLogger.addEntry method - if the total number of active entrylogs is more than or equal to number of active ledgers, then we get almost exclusivity - For implementing Round-Robin approach, we need to maintain state information of mapping of ledgerId to SlotId - there will be numberofledgerdirs*numberofactiveentrylogsperledgerdir slots. a slot is mapped to ledgerdir, but the activeentrylog of that slot will be rotated when it reaches the capacity. - By knowing the SlotId we can get the corresponding entryLogId associated to that slot. - If there is no entry for current ledger in the map, then we pick the next in order slot and add the mapping entry to the map. - Since Bookie won't be informed about the writeclose of the ledger, there is no easy way to know when to remove the mapping entry from the map. Considering it is just <long ledgerid, int slotid> mapentry, we may compromise on evicting policy. We can just use some Cache, which has eviction policy, timebased on last access - If a ledgerdir becomes full, then all the slots having entrylogs in that ledgerdir, should become inactive. The existing mappings, mappings of active ledgers to these slots (active entrylogs), should be updated to available active slots. - when ledgerdir becomes writable again, then the slots which were inactive should be made active and become eligible for round-robin distribution - For this feature I need to make changes to checkpoint logic. Currently with BOOKKEEPER-564 change, we are scheduling checkpoint only when current entrylog file is rotated. So we dont call 'flushCurrentLog' when we checkpoint. But for this feature, since there are going to be multiple active entrylogs, scheduling checkpoint when entrylog file is rotated, is not an option. So I need to call flushCurrentLogs when checkpoint is made for every 'flushinterval' period --- *Enrico Olivelli* 2017-05-19T06:49:35.104+0000 [~jujjuri] [~reddycharan18@gmail.com] This sound very interesting. Now I can see clearly way JV wrote on the mailing list that maybe clients could send some hint to the bookies that a ledger has been gracefully deleted/closed --- *Charan Reddy Guttapalem* 2017-06-02T00:22:59.045+0000 [~sijie@apache.org] created writeup for this work item and discussed about it in last community call (May 18th)
label: code-design

2. **title:** Multiple active entrylogs
body: JIRA: https://issues.apache.org/jira/browse/BOOKKEEPER-1041 Reporter: Venkateswararao Jujjuri (JV) @jvrao Current bookkeeper is tuned for rotational HDDs. It has one active entrylog, and all the ledger/entries go to the same entrylog until it is rotated out. This is perfect for HDDs as seeks and moving head allover the disk platter is very expensive. But this is very inefficient for SSDs, as each SSD can handle multiple parallel writers, also this method is extremely inefficient for compaction as it causes write amplification and inefficient disk space usage. Our proposal is to have multiple active entrylogs and a configuration param on how many parallel entrylogs the system can have. This way one can have ability to configure to have less (may be one) ledger per entrylog. ### Comments from JIRA --- *Enrico Olivelli* 2017-04-20T07:28:28.619+0000 {quote} But this is very inefficient for HDDs {quote} [~jujjuri] did you mean SSD ? --- *Venkateswararao Jujjuri (JV)* 2017-04-21T16:42:29.203+0000 Yes I mean SSD. Corrected. Thanks. --- *Charan Reddy Guttapalem* 2017-05-18T15:01:22.566+0000 Issue: In Bookie's EntryLogger, we are having only one current active entryLog and all the ledger/entries go to the same entrylog. This is perfect for HDDs as file syncs, seeks and moving head allover the disk platter is very expensive. But having single active Entry Log is inefficient for SSDs, as each SSD can handle multiple parallel writers. Also, having single active EntryLog (irrespective of LedgerStorage type - interleaved/sorted), is inefficient for compaction, since entries of multiple ledgers will end up in an entrylog. Also in SortedLedgerStorage , in the addEntry request we flush EntryMemtable, if it reaches the sizelimit. Because of this we are observing unpredictable tail latency for addEntry request. When EntryMemTable snapshot of size (64 MB) is flushed all at once, this may affect the journal addentry latency. Also, if the rate of new add requests surpasses the rate at which the EntryMemTable's previous snapshot is flushed, then at a point the current EntryMemTable map will reach the limit and since the previous snapshot flush is in progress, EntryMemTable will throttle new addRequests, which would affect addEntry latency. Goals: The main purpose of this feature is to have efficient Garbagecollection story by minimizing the amount of compactions required and the ability to reclaim the deleted ledger's space quicker. Also with this feature we can lay foreground for switching to InterleavedLedgerStorage from SortedLedgerStorage and get predictable tail latency. Proposal: So proposal here is to have multiple active entrylogs. Which will help with compaction performance and make SortedLedgerStorage redundant. Design Overview: - is to have server configuration specifying number of active entry logs per ledgerdir. - for backward compatibility (for existing behaviour) that config can be set to 0. - round-robin method will be used for choosing the active entry log for the current ledger in EntryLogger.addEntry method - if the total number of active entrylogs is more than or equal to number of active ledgers, then we get almost exclusivity - For implementing Round-Robin approach, we need to maintain state information of mapping of ledgerId to SlotId - there will be numberofledgerdirs*numberofactiveentrylogsperledgerdir slots. a slot is mapped to ledgerdir, but the activeentrylog of that slot will be rotated when it reaches the capacity. - By knowing the SlotId we can get the corresponding entryLogId associated to that slot. - If there is no entry for current ledger in the map, then we pick the next in order slot and add the mapping entry to the map. - Since Bookie won't be informed about the writeclose of the ledger, there is no easy way to know when to remove the mapping entry from the map. Considering it is just <long ledgerid, int slotid> mapentry, we may compromise on evicting policy. We can just use some Cache, which has eviction policy, timebased on last access - If a ledgerdir becomes full, then all the slots having entrylogs in that ledgerdir, should become inactive. The existing mappings, mappings of active ledgers to these slots (active entrylogs), should be updated to available active slots. - when ledgerdir becomes writable again, then the slots which were inactive should be made active and become eligible for round-robin distribution - For this feature I need to make changes to checkpoint logic. Currently with BOOKKEEPER-564 change, we are scheduling checkpoint only when current entrylog file is rotated. So we dont call 'flushCurrentLog' when we checkpoint. But for this feature, since there are going to be multiple active entrylogs, scheduling checkpoint when entrylog file is rotated, is not an option. So I need to call flushCurrentLogs when checkpoint is made for every 'flushinterval' period --- *Enrico Olivelli* 2017-05-19T06:49:35.104+0000 [~jujjuri] [~reddycharan18@gmail.com] This sound very interesting. Now I can see clearly way JV wrote on the

mailing list that maybe clients could send some hint to the bookies that a ledger has been gracefully deleted/closed --- *Charan Reddy Guttapalem* 2017-06-02T00:22:59.045+0000 [~sijie@apache.org] created writeup for this work item and discussed about it in last community call (May 18th)

3. **title:** Multiple active entrylogs
   body: JIRA: https://issues.apache.org/jira/browse/BOOKKEEPER-1041 Reporter: Venkateswararao Jujjuri (JV) @jvrao Current bookkeeper is tuned for rotational HDDs. It has one active entrylog, and all the ledger/entries go to the same entrylog until it is rotated out. This is perfect for HDDs as seeks and moving head allover the disk platter is very expensive. But this is very inefficient for SSDs, as each SSD can handle multiple parallel writers, also this method is extremely inefficient for compaction as it causes write amplification and inefficient disk space usage. Our proposal is to have multiple active entrylogs and a configuration param on how many parallel entrylogs the system can have. This way one can have ability to configure to have less (may be one) ledger per entrylog. ### Comments from JIRA --- *Enrico Olivelli* 2017-04-20T07:28:28.619+0000 {quote} But this is very inefficient for HDDs {quote} [~jujjuri] did you mean SSD ? --- *Venkateswararao Jujjuri (JV)* 2017-04-21T16:42:29.203+0000 Yes I mean SSD. Corrected. Thanks. --- *Charan Reddy Guttapalem* 2017-05-18T15:01:22.566+0000 Issue: In Bookie's EntryLogger, we are having only one current active entryLog and all the ledger/entries go to the same entrylog. This is perfect for HDDs as file syncs, seeks and moving head allover the disk platter is very expensive. But having single active Entry Log is inefficient for SSDs, as each SSD can handle multiple parallel writers. Also, having single active EntryLog (irrespective of LedgerStorage type - interleaved/sorted), is inefficient for compaction, since entries of multiple ledgers will end up in an entrylog. Also in SortedLedgerStorage , in the addEntry request we flush EntryMemtable, if it reaches the sizelimit. Because of this we are observing unpredictable tail latency for addEntry request. When EntryMemTable snapshot of size (64 MB) is flushed all at once, this may affect the journal addentry latency. Also, if the rate of new add requests surpasses the rate at which the EntryMemTable's previous snapshot is flushed, then at a point the current EntryMemTable map will reach the limit and since the previous snapshot flush is in progress, EntryMemTable will throttle new addRequests, which would affect addEntry latency. Goals: The main purpose of this feature is to have efficient Garbagecollection story by minimizing the amount of compactions required and the ability to reclaim the deleted ledger's space quicker. Also with this feature we can lay foreground for switching to InterleavedLedgerStorage from SortedLedgerStorage and get predictable tail latency. Proposal: So proposal here is to have multiple active entrylogs. Which will help with compaction performance and make SortedLedgerStorage redundant. Design Overview: - is to have server configuration specifying number of active entry logs per ledgerdir. - for backward compatibility (for existing behaviour) that config can be set to 0. - round-robin method will be used for choosing the active entry log for the current ledger in EntryLogger.addEntry method - if the total number of active entrylogs is more than or equal to number of active ledgers, then we get almost exclusivity - For implementing Round-Robin approach, we need to maintain state information of mapping of ledgerId to SlotId - there will be numberofledgerdirs*numberofactiveentrylogsperledgerdir slots. a slot is mapped to ledgerdir, but the activeentrylog of that slot will be rotated when it reaches the capacity. - By knowing the SlotId we can get the corresponding entryLogId associated to that slot. - If there is no entry for current ledger in the map, then we pick the next in order slot and add the mapping entry to the map. - Since Bookie won't be informed about the writeclose of the ledger, there is no easy way to know when to remove the mapping entry from the map. Considering it is just <long ledgerid, int slotid> mapentry, we may compromise on evicting policy. We can just use some Cache, which has eviction policy, timebased on last access - If a ledgerdir becomes full, then all the slots having entrylogs in that ledgerdir, should become inactive. The existing mappings, mappings of active ledgers to these slots (active entrylogs), should be updated to available active slots. - when ledgerdir becomes writable again, then the slots which were inactive should be made active and become eligible for round-robin distribution - For this feature I need to make changes to checkpoint logic. Currently with BOOKKEEPER-564 change, we are scheduling checkpoint only when current entrylog file is rotated. So we dont call 'flushCurrentLog' when we checkpoint. But for this feature, since there are going to be multiple active entrylogs, scheduling checkpoint when entrylog file is rotated, is not an option. So I need to call flushCurrentLogs when checkpoint is made for every 'flushinterval' period --- *Enrico Olivelli* 2017-05-19T06:49:35.104+0000 [~jujjuri] [~reddycharan18@gmail.com] This sound very interesting. Now I can see clearly way JV wrote on the mailing list that maybe clients could send some hint to the bookies that a ledger has been gracefully deleted/closed --- *Charan Reddy Guttapalem* 2017-06-02T00:22:59.045+0000 [~sijie@apache.org] created writeup for this work item and discussed about it in last community call (May 18th)

4. **title:** Multiple active entrylogs
   body: JIRA: https://issues.apache.org/jira/browse/BOOKKEEPER-1041 Reporter: Venkateswararao Jujjuri (JV) @jvrao Current bookkeeper is tuned for rotational HDDs. It has one active entrylog, and all the ledger/entries go to the same entrylog until it is rotated out. This is perfect for HDDs as seeks and moving head allover the disk platter is very expensive. But this is very inefficient for SSDs, as each SSD can handle multiple parallel writers, also this method is extremely inefficient for compaction as it causes write amplification and inefficient disk space usage. Our proposal is to have multiple active entrylogs and a configuration param on how many parallel entrylogs the system can have. This way one can have ability to configure to have less (may be one) ledger per entrylog. ### Comments from JIRA --- *Enrico Olivelli* 2017-04-20T07:28:28.619+0000 {quote} But this is very inefficient for HDDs {quote} [~jujjuri] did you mean SSD ? --- *Venkateswararao Jujjuri (JV)* 2017-04-21T16:42:29.203+0000 Yes I mean SSD. Corrected. Thanks. --- *Charan Reddy Guttapalem* 2017-05-18T15:01:22.566+0000 Issue: In Bookie's EntryLogger, we are having only one current active entryLog and all the ledger/entries go to the same entrylog. This is perfect for HDDs as file syncs, seeks and moving head allover the disk platter is very expensive. But having single active Entry Log is inefficient for SSDs, as each SSD can handle multiple parallel writers. Also, having single active EntryLog (irrespective of LedgerStorage type - interleaved/sorted), is inefficient for compaction, since entries of multiple ledgers will end up in an entrylog. Also in SortedLedgerStorage , in the addEntry request we flush EntryMemtable, if it reaches the sizelimit. Because of this we are observing unpredictable tail latency for addEntry request. When EntryMemTable snapshot of size (64 MB) is flushed all at once, this may affect the journal addentry latency. Also, if the rate of new add requests surpasses the rate at which the EntryMemTable's previous snapshot is flushed, then at a point the current EntryMemTable map will reach the limit and since the previous snapshot flush is in progress, EntryMemTable will throttle new addRequests, which would affect addEntry latency. Goals: The main purpose of this feature is to have efficient Garbagecollection story by minimizing the amount of compactions required and the ability to reclaim the deleted ledger's space quicker. Also with this feature we can lay foreground for switching to InterleavedLedgerStorage from SortedLedgerStorage and get predictable tail latency. Proposal: So proposal here is to have multiple active entrylogs. Which will help with compaction performance and make SortedLedgerStorage redundant. Design Overview: - is to have server configuration specifying number of active entry logs per ledgerdir. - for backward compatibility (for existing behaviour) that config can be set to 0. - round-robin method will be used for choosing the active entry log for the current ledger in EntryLogger.addEntry method - if the total number of active entrylogs is more than or equal to number of active ledgers, then we get almost exclusivity - For implementing Round-Robin approach, we need to maintain state information of mapping of ledgerId to SlotId - there will be numberofledgerdirs*numberofactiveentrylogsperledgerdir slots. a slot is mapped to ledgerdir, but the activeentrylog of that slot will be rotated when it reaches the capacity. - By knowing the SlotId we can get the corresponding entryLogId associated to that slot. - If there is no entry for current ledger in the map, then we pick the next in order slot and add the mapping entry to the map. - Since Bookie won't be informed about the writeclose of the ledger, there is no easy way to know when to remove the mapping entry from the map. Considering it is just <long ledgerid, int slotid> mapentry, we may compromise on evicting policy. We can just use some Cache, which has eviction policy, timebased on last access - If a ledgerdir becomes full, then all the slots having entrylogs in that ledgerdir, should become inactive. The existing mappings, mappings of active ledgers to these slots (active entrylogs), should be updated to available active slots. - when ledgerdir becomes writable again, then the slots which were inactive should be made active and become

eligible for round-robin distribution - For this feature I need to make changes to checkpoint logic. Currently with BOOKKEEPER-564 change, we are scheduling checkpoint only when current entrylog file is rotated. So we dont call 'flushCurrentLog' when we checkpoint. But for this feature, since there are going to be multiple active entrylogs, scheduling checkpoint when entrylog file is rotated, is not an option. So I need to call flushCurrentLogs when checkpoint is made for every 'flushinterval' period --- *Enrico Olivelli* 2017-05-19T06:49:35.104+0000 [~jujjuri] [~reddycharan18@gmail.com] This sound very interesting. Now I can see clearly way JV wrote on the mailing list that maybe clients could send some hint to the bookies that a ledger has been gracefully deleted/closed --- *Charan Reddy Guttapalem* 2017-06-02T00:22:59.045+0000 [~sijie@apache.org] created writeup for this work item and discussed about it in last community call (May 18th)
label: code-design

5. **title:** Multiple active entrylogs
body: JIRA: https://issues.apache.org/jira/browse/BOOKKEEPER-1041 Reporter: Venkateswararao Jujjuri (JV) @jvrao Current bookkeeper is tuned for rotational HDDs. It has one active entrylog, and all the ledger/entries go to the same entrylog until it is rotated out. This is perfect for HDDs as seeks and moving head allover the disk platter is very expensive. But this is very inefficient for SSDs, as each SSD can handle multiple parallel writers, also this method is extremely inefficient for compaction as it causes write amplification and inefficient disk space usage. Our proposal is to have multiple active entrylogs and a configuration param on how many parallel entrylogs the system can have. This way one can have ability to configure to have less (may be one) ledger per entrylog. ### Comments from JIRA --- *Enrico Olivelli* 2017-04-20T07:28:28.619+0000 {quote} But this is very inefficient for HDDs {quote} [~jujjuri] did you mean SSD ? --- *Venkateswararao Jujjuri (JV)* 2017-04-21T16:42:29.203+0000 Yes I mean SSD. Corrected. Thanks. --- *Charan Reddy Guttapalem* 2017-05-18T15:01:22.566+0000 Issue: In Bookie's EntryLogger, we are having only one current active entryLog and all the ledger/entries go to the same entrylog. This is perfect for HDDs as file syncs, seeks and moving head allover the disk platter is very expensive. But having single active Entry Log is inefficient for SSDs, as each SSD can handle multiple parallel writers. Also, having single active EntryLog (irrespective of LedgerStorage type - interleaved/sorted), is inefficient for compaction, since entries of multiple ledgers will end up in an entrylog. Also in SortedLedgerStorage , in the addEntry request we flush EntryMemtable, if it reaches the sizelimit. Because of this we are observing unpredictable tail latency for addEntry request. When EntryMemTable snapshot of size (64 MB) is flushed all at once, this may affect the journal addentry latency. Also, if the rate of new add requests surpasses the rate at which the EntryMemTable's previous snapshot is flushed, then at a point the current EntryMemTable map will reach the limit and since the previous snapshot flush is in progress, EntryMemTable will throttle new addRequests, which would affect addEntry latency. Goals: The main purpose of this feature is to have efficient Garbagecollection story by minimizing the amount of compactions required and the ability to reclaim the deleted ledger's space quicker. Also with this feature we can lay foreground for switching to InterleavedLedgerStorage from SortedLedgerStorage and get predictable tail latency. Proposal: So proposal here is to have multiple active entrylogs. Which will help with compaction performance and make SortedLedgerStorage redundant. Design Overview: - is to have server configuration specifying number of active entry logs per ledgerdir. - for backward compatibility (for existing behaviour) that config can be set to 0. - round-robin method will be used for choosing the active entry log for the current ledger in EntryLogger.addEntry method - if the total number of active entrylogs is more than or equal to number of active ledgers, then we get almost exclusivity - For implementing Round-Robin approach, we need to maintain state information of mapping of ledgerId to SlotId - there will be numberofledgerdirs*numberofactiveentrylogsperledgerdir slots. a slot is mapped to ledgerdir, but the activeentrylog of that slot will be rotated when it reaches the capacity. - By knowing the SlotId we can get the corresponding entryLogId associated to that slot. - If there is no entry for current ledger in the map, then we pick the next in order slot and add the mapping entry to the map. - Since Bookie won't be informed about the writeclose of the ledger, there is no easy way to know when to remove the mapping entry from the map. Considering it is just <long ledgerid, int slotid> mapentry, we may compromise on evicting policy. We can just use some Cache, which has eviction policy, timebased on last access - If a ledgerdir becomes full, then all the slots having entrylogs in that ledgerdir, should become inactive. The existing mappings, mappings of active ledgers to these slots (active entrylogs), should be updated to available active slots. - when ledgerdir becomes writable again, then the slots which were inactive should be made active and become eligible for round-robin distribution - For this feature I need to make changes to checkpoint logic. Currently with BOOKKEEPER-564 change, we are scheduling checkpoint only when current entrylog file is rotated. So we dont call 'flushCurrentLog' when we checkpoint. But for this feature, since there are going to be multiple active entrylogs, scheduling checkpoint when entrylog file is rotated, is not an option. So I need to call flushCurrentLogs when checkpoint is made for every 'flushinterval' period --- *Enrico Olivelli* 2017-05-19T06:49:35.104+0000 [~jujjuri] [~reddycharan18@gmail.com] This sound very interesting. Now I can see clearly way JV wrote on the mailing list that maybe clients could send some hint to the bookies that a ledger has been gracefully deleted/closed --- *Charan Reddy Guttapalem* 2017-06-02T00:22:59.045+0000 [~sijie@apache.org] created writeup for this work item and discussed about it in last community call (May 18th)
label: code-design

6. **title:** Multiple active entrylogs
body: JIRA: https://issues.apache.org/jira/browse/BOOKKEEPER-1041 Reporter: Venkateswararao Jujjuri (JV) @jvrao Current bookkeeper is tuned for rotational HDDs. It has one active entrylog, and all the ledger/entries go to the same entrylog until it is rotated out. This is perfect for HDDs as seeks and moving head allover the disk platter is very expensive. But this is very inefficient for SSDs, as each SSD can handle multiple parallel writers, also this method is extremely inefficient for compaction as it causes write amplification and inefficient disk space usage. Our proposal is to have multiple active entrylogs and a configuration param on how many parallel entrylogs the system can have. This way one can have ability to configure to have less (may be one) ledger per entrylog. ### Comments from JIRA --- *Enrico Olivelli* 2017-04-20T07:28:28.619+0000 {quote} But this is very inefficient for HDDs {quote} [~jujjuri] did you mean SSD ? --- *Venkateswararao Jujjuri (JV)* 2017-04-21T16:42:29.203+0000 Yes I mean SSD. Corrected. Thanks. --- *Charan Reddy Guttapalem* 2017-05-18T15:01:22.566+0000 Issue: In Bookie's EntryLogger, we are having only one current active entryLog and all the ledger/entries go to the same entrylog. This is perfect for HDDs as file syncs, seeks and moving head allover the disk platter is very expensive. But having single active Entry Log is inefficient for SSDs, as each SSD can handle multiple parallel writers. Also, having single active EntryLog (irrespective of LedgerStorage type - interleaved/sorted), is inefficient for compaction, since entries of multiple ledgers will end up in an entrylog. Also in SortedLedgerStorage , in the addEntry request we flush EntryMemtable, if it reaches the sizelimit. Because of this we are observing unpredictable tail latency for addEntry request. When EntryMemTable snapshot of size (64 MB) is flushed all at once, this may affect the journal addentry latency. Also, if the rate of new add requests surpasses the rate at which the EntryMemTable's previous snapshot is flushed, then at a point the current EntryMemTable map will reach the limit and since the previous snapshot flush is in progress, EntryMemTable will throttle new addRequests, which would affect addEntry latency. Goals: The main purpose of this feature is to have efficient Garbagecollection story by minimizing the amount of compactions required and the ability to reclaim the deleted ledger's space quicker. Also with this feature we can lay foreground for switching to InterleavedLedgerStorage from SortedLedgerStorage and get predictable tail latency. Proposal: So proposal here is to have multiple active entrylogs. Which will help with compaction performance and make SortedLedgerStorage redundant. Design Overview: - is to have server configuration specifying number of active entry logs per ledgerdir. - for backward compatibility (for existing behaviour) that config can be set to 0. - round-robin method will be used for choosing the active entry log for the current ledger in EntryLogger.addEntry method - if the total number of active entrylogs is more than or equal to number of active ledgers, then we get almost exclusivity - For implementing Round-Robin approach, we need to maintain state information of mapping of ledgerId to SlotId - there will be numberofledgerdirs*numberofactiveentrylogsperledgerdir slots. a slot is mapped to ledgerdir,

but the activeentrylog of that slot will be rotated when it reaches the capacity. - By knowing the SlotId we can get the corresponding entryLogId associated to that slot. - If there is no entry for current ledger in the map, then we pick the next in order slot and add the mapping entry to the map. - Since Bookie won't be informed about the writeclose of the ledger, there is no easy way to know when to remove the mapping entry from the map. Considering it is just <long ledgerid, int slotid> mapentry, we may compromise on evicting policy. We can just use some Cache, which has eviction policy, timebased on last access - If a ledgerdir becomes full, then all the slots having entrylogs in that ledgerdir, should become inactive. The existing mappings, mappings of active ledgers to these slots (active entrylogs), should be updated to available active slots. - when ledgerdir becomes writable again, then the slots which were inactive should be made active and become eligible for round-robin distribution - For this feature I need to make changes to checkpoint logic. Currently with BOOKKEEPER-564 change, we are scheduling checkpoint only when current entrylog file is rotated. So we dont call 'flushCurrentLog' when we checkpoint. But for this feature, since there are going to be multiple active entrylogs, scheduling checkpoint when entrylog file is rotated, is not an option. So I need to call flushCurrentLogs when checkpoint is made for every 'flushinterval' period --- *Enrico Olivelli* 2017-05-19T06:49:35.104+0000 [~jujjuri] [~reddycharan18@gmail.com] This sound very interesting. Now I can see clearly way JV wrote on the mailing list that maybe clients could send some hint to the bookies that a ledger has been gracefully deleted/closed --- *Charan Reddy Guttapalem* 2017-06-02T00:22:59.045+0000 [~sijie@apache.org] created writeup for this work item and discussed about it in last community call (May 18th)
label: code-design

7. **title:** Multiple active entrylogs
body: JIRA: https://issues.apache.org/jira/browse/BOOKKEEPER-1041 Reporter: Venkateswararao Jujjuri (JV) @jvrao Current bookkeeper is tuned for rotational HDDs. It has one active entrylog, and all the ledger/entries go to the same entrylog until it is rotated out. This is perfect for HDDs as seeks and moving head allover the disk platter is very expensive. But this is very inefficient for SSDs, as each SSD can handle multiple parallel writers, also this method is extremely inefficient for compaction as it causes write amplification and inefficient disk space usage. Our proposal is to have multiple active entrylogs and a configuration param on how many parallel entrylogs the system can have. This way one can have ability to configure to have less (may be one) ledger per entrylog. ### Comments from JIRA --- *Enrico Olivelli* 2017-04-20T07:28:28.619+0000 {quote} But this is very inefficient for HDDs {quote} [~jujjuri] did you mean SSD ? --- *Venkateswararao Jujjuri (JV)* 2017-04-21T16:42:29.203+0000 Yes I mean SSD. Corrected. Thanks. --- *Charan Reddy Guttapalem* 2017-05-18T15:01:22.566+0000 Issue: In Bookie's EntryLogger, we are having only one current active entryLog and all the ledger/entries go to the same entrylog. This is perfect for HDDs as file syncs, seeks and moving head allover the disk platter is very expensive. But having single active Entry Log is inefficient for SSDs, as each SSD can handle multiple parallel writers. Also, having single active EntryLog (irrespective of LedgerStorage type - interleaved/sorted), is inefficient for compaction, since entries of multiple ledgers will end up in an entrylog. Also in SortedLedgerStorage , in the addEntry request we flush EntryMemtable, if it reaches the sizelimit. Because of this we are observing unpredictable tail latency for addEntry request. When EntryMemTable snapshot of size (64 MB) is flushed all at once, this may affect the journal addentry latency. Also, if the rate of new add requests surpasses the rate at which the EntryMemTable's previous snapshot is flushed, then at a point the current EntryMemTable map will reach the limit and since the previous snapshot flush is in progress, EntryMemTable will throttle new addRequests, which would affect addEntry latency. Goals: The main purpose of this feature is to have efficient Garbagecollection story by minimizing the amount of compactions required and the ability to reclaim the deleted ledger's space quicker. Also with this feature we can lay foreground for switching to InterleavedLedgerStorage from SortedLedgerStorage and get predictable tail latency. Proposal: So proposal here is to have multiple active entrylogs. Which will help with compaction performance and make SortedLedgerStorage redundant. Design Overview: - is to have server configuration specifying number of active entry logs per ledgerdir. - for backward compatibility (for existing behaviour) that config can be set to 0. - round-robin method will be used for choosing the active entry log for the current ledger in EntryLogger.addEntry method - if the total number of active entrylogs is more than or equal to number of active ledgers, then we get almost exclusivity - For implementing Round-Robin approach, we need to maintain state information of mapping of ledgerId to SlotId - there will be numberofledgerdirs*numberofactiveentrylogsperledgerdir slots. a slot is mapped to ledgerdir, but the activeentrylog of that slot will be rotated when it reaches the capacity. - By knowing the SlotId we can get the corresponding entryLogId associated to that slot. - If there is no entry for current ledger in the map, then we pick the next in order slot and add the mapping entry to the map. - Since Bookie won't be informed about the writeclose of the ledger, there is no easy way to know when to remove the mapping entry from the map. Considering it is just <long ledgerid, int slotid> mapentry, we may compromise on evicting policy. We can just use some Cache, which has eviction policy, timebased on last access - If a ledgerdir becomes full, then all the slots having entrylogs in that ledgerdir, should become inactive. The existing mappings, mappings of active ledgers to these slots (active entrylogs), should be updated to available active slots. - when ledgerdir becomes writable again, then the slots which were inactive should be made active and become eligible for round-robin distribution - For this feature I need to make changes to checkpoint logic. Currently with BOOKKEEPER-564 change, we are scheduling checkpoint only when current entrylog file is rotated. So we dont call 'flushCurrentLog' when we checkpoint. But for this feature, since there are going to be multiple active entrylogs, scheduling checkpoint when entrylog file is rotated, is not an option. So I need to call flushCurrentLogs when checkpoint is made for every 'flushinterval' period --- *Enrico Olivelli* 2017-05-19T06:49:35.104+0000 [~jujjuri] [~reddycharan18@gmail.com] This sound very interesting. Now I can see clearly way JV wrote on the mailing list that maybe clients could send some hint to the bookies that a ledger has been gracefully deleted/closed --- *Charan Reddy Guttapalem* 2017-06-02T00:22:59.045+0000 [~sijie@apache.org] created writeup for this work item and discussed about it in last community call (May 18th)

8. **title:** Multiple active entrylogs
body: JIRA: https://issues.apache.org/jira/browse/BOOKKEEPER-1041 Reporter: Venkateswararao Jujjuri (JV) @jvrao Current bookkeeper is tuned for rotational HDDs. It has one active entrylog, and all the ledger/entries go to the same entrylog until it is rotated out. This is perfect for HDDs as seeks and moving head allover the disk platter is very expensive. But this is very inefficient for SSDs, as each SSD can handle multiple parallel writers, also this method is extremely inefficient for compaction as it causes write amplification and inefficient disk space usage. Our proposal is to have multiple active entrylogs and a configuration param on how many parallel entrylogs the system can have. This way one can have ability to configure to have less (may be one) ledger per entrylog. ### Comments from JIRA --- *Enrico Olivelli* 2017-04-20T07:28:28.619+0000 {quote} But this is very inefficient for HDDs {quote} [~jujjuri] did you mean SSD ? --- *Venkateswararao Jujjuri (JV)* 2017-04-21T16:42:29.203+0000 Yes I mean SSD. Corrected. Thanks. --- *Charan Reddy Guttapalem* 2017-05-18T15:01:22.566+0000 Issue: In Bookie's EntryLogger, we are having only one current active entryLog and all the ledger/entries go to the same entrylog. This is perfect for HDDs as file syncs, seeks and moving head allover the disk platter is very expensive. But having single active Entry Log is inefficient for SSDs, as each SSD can handle multiple parallel writers. Also, having single active EntryLog (irrespective of LedgerStorage type - interleaved/sorted), is inefficient for compaction, since entries of multiple ledgers will end up in an entrylog. Also in SortedLedgerStorage , in the addEntry request we flush EntryMemtable, if it reaches the sizelimit. Because of this we are observing unpredictable tail latency for addEntry request. When EntryMemTable snapshot of size (64 MB) is flushed all at once, this may affect the journal addentry latency. Also, if the rate of new add requests surpasses the rate at which the EntryMemTable's previous snapshot is flushed, then at a point the current EntryMemTable map will reach the limit and since the previous snapshot flush is in progress, EntryMemTable will throttle new addRequests, which would affect addEntry latency. Goals: The main purpose of this feature is to have efficient Garbagecollection story by minimizing the amount of compactions required and the ability to reclaim the deleted ledger's space quicker. Also with this feature we can lay foreground for switching to InterleavedLedgerStorage from SortedLedgerStorage and get

predictable tail latency. Proposal: So proposal here is to have multiple active entrylogs. Which will help with compaction performance and make SortedLedgerStorage redundant. Design Overview: - is to have server configuration specifying number of active entry logs per ledgerdir. - for backward compatibility (for existing behaviour) that config can be set to 0. - round-robin method will be used for choosing the active entry log for the current ledger in EntryLogger.addEntry method - if the total number of active entrylogs is more than or equal to number of active ledgers, then we get almost exclusivity - For implementing Round-Robin approach, we need to maintain state information of mapping of ledgerId to SlotId - there will be numberofledgerdirs*numberofactiveentrylogsperledgerdir slots. a slot is mapped to ledgerdir, but the activeentrylog of that slot will be rotated when it reaches the capacity. - By knowing the SlotId we can get the corresponding entryLogId associated to that slot. - If there is no entry for current ledger in the map, then we pick the next in order slot and add the mapping entry to the map. - Since Bookie won't be informed about the writeclose of the ledger, there is no easy way to know when to remove the mapping entry from the map. Considering it is just <long ledgerid, int slotid> mapentry, we may compromise on evicting policy. We can just use some Cache, which has eviction policy, timebased on last access - If a ledgerdir becomes full, then all the slots having entrylogs in that ledgerdir, should become inactive. The existing mappings, mappings of active ledgers to these slots (active entrylogs), should be updated to available active slots. - when ledgerdir becomes writable again, then the slots which were inactive should be made active and become eligible for round-robin distribution - For this feature I need to make changes to checkpoint logic. Currently with BOOKKEEPER-564 change, we are scheduling checkpoint only when current entrylog file is rotated. So we dont call 'flushCurrentLog' when we checkpoint. But for this feature, since there are going to be multiple active entrylogs, scheduling checkpoint when entrylog file is rotated, is not an option. So I need to call flushCurrentLogs when checkpoint is made for every 'flushinterval' period --- *Enrico Olivelli* 2017-05-19T06:49:35.104+0000 [~jujjuri] [~reddycharan18@gmail.com] This sound very interesting. Now I can see clearly way JV wrote on the mailing list that maybe clients could send some hint to the bookies that a ledger has been gracefully deleted/closed --- *Charan Reddy Guttapalem* 2017-06-02T00:22:59.045+0000 [~sijie@apache.org] created writeup for this work item and discussed about it in last community call (May 18th)

9. **title:** Multiple active entrylogs
   body: JIRA: https://issues.apache.org/jira/browse/BOOKKEEPER-1041 Reporter: Venkateswararao Jujjuri (JV) @jvrao Current bookkeeper is tuned for rotational HDDs. It has one active entrylog, and all the ledger/entries go to the same entrylog until it is rotated out. This is perfect for HDDs as seeks and moving head allover the disk platter is very expensive. But this is very inefficient for SSDs, as each SSD can handle multiple parallel writers, also this method is extremely inefficient for compaction as it causes write amplification and inefficient disk space usage. Our proposal is to have multiple active entrylogs and a configuration param on how many parallel entrylogs the system can have. This way one can have ability to configure to have less (may be one) ledger per entrylog. ### Comments from JIRA --- *Enrico Olivelli* 2017-04-20T07:28:28.619+0000 {quote} But this is very inefficient for HDDs {quote} [~jujjuri] did you mean SSD ? --- *Venkateswararao Jujjuri (JV)* 2017-04-21T16:42:29.203+0000 Yes I mean SSD. Corrected. Thanks. --- *Charan Reddy Guttapalem* 2017-05-18T15:01:22.566+0000 Issue: In Bookie's EntryLogger, we are having only one current active entryLog and all the ledger/entries go to the same entrylog. This is perfect for HDDs as file syncs, seeks and moving head allover the disk platter is very expensive. But having single active Entry Log is inefficient for SSDs, as each SSD can handle multiple parallel writers. Also, having single active EntryLog (irrespective of LedgerStorage type - interleaved/sorted), is inefficient for compaction, since entries of multiple ledgers will end up in an entrylog. Also in SortedLedgerStorage , in the addEntry request we flush EntryMemtable, if it reaches the sizelimit. Because of this we are observing unpredictable tail latency for addEntry request. When EntryMemTable snapshot of size (64 MB) is flushed all at once, this may affect the journal addentry latency. Also, if the rate of new add requests surpasses the rate at which the EntryMemTable's previous snapshot is flushed, then at a point the current EntryMemTable map will reach the limit and since the previous snapshot flush is in progress, EntryMemTable will throttle new addRequests, which would affect addEntry latency. Goals: The main purpose of this feature is to have efficient Garbagecollection story by minimizing the amount of compactions required and the ability to reclaim the deleted ledger's space quicker. Also with this feature we can lay foreground for switching to InterleavedLedgerStorage from SortedLedgerStorage and get predictable tail latency. Proposal: So proposal here is to have multiple active entrylogs. Which will help with compaction performance and make SortedLedgerStorage redundant. Design Overview: - is to have server configuration specifying number of active entry logs per ledgerdir. - for backward compatibility (for existing behaviour) that config can be set to 0. - round-robin method will be used for choosing the active entry log for the current ledger in EntryLogger.addEntry method - if the total number of active entrylogs is more than or equal to number of active ledgers, then we get almost exclusivity - For implementing Round-Robin approach, we need to maintain state information of mapping of ledgerId to SlotId - there will be numberofledgerdirs*numberofactiveentrylogsperledgerdir slots. a slot is mapped to ledgerdir, but the activeentrylog of that slot will be rotated when it reaches the capacity. - By knowing the SlotId we can get the corresponding entryLogId associated to that slot. - If there is no entry for current ledger in the map, then we pick the next in order slot and add the mapping entry to the map. - Since Bookie won't be informed about the writeclose of the ledger, there is no easy way to know when to remove the mapping entry from the map. Considering it is just <long ledgerid, int slotid> mapentry, we may compromise on evicting policy. We can just use some Cache, which has eviction policy, timebased on last access - If a ledgerdir becomes full, then all the slots having entrylogs in that ledgerdir, should become inactive. The existing mappings, mappings of active ledgers to these slots (active entrylogs), should be updated to available active slots. - when ledgerdir becomes writable again, then the slots which were inactive should be made active and become eligible for round-robin distribution - For this feature I need to make changes to checkpoint logic. Currently with BOOKKEEPER-564 change, we are scheduling checkpoint only when current entrylog file is rotated. So we dont call 'flushCurrentLog' when we checkpoint. But for this feature, since there are going to be multiple active entrylogs, scheduling checkpoint when entrylog file is rotated, is not an option. So I need to call flushCurrentLogs when checkpoint is made for every 'flushinterval' period --- *Enrico Olivelli* 2017-05-19T06:49:35.104+0000 [~jujjuri] [~reddycharan18@gmail.com] This sound very interesting. Now I can see clearly way JV wrote on the mailing list that maybe clients could send some hint to the bookies that a ledger has been gracefully deleted/closed --- *Charan Reddy Guttapalem* 2017-06-02T00:22:59.045+0000 [~sijie@apache.org] created writeup for this work item and discussed about it in last community call (May 18th)
   label: code-design
10. **title:** Multiple active entrylogs
    body: JIRA: https://issues.apache.org/jira/browse/BOOKKEEPER-1041 Reporter: Venkateswararao Jujjuri (JV) @jvrao Current bookkeeper is tuned for rotational HDDs. It has one active entrylog, and all the ledger/entries go to the same entrylog until it is rotated out. This is perfect for HDDs as seeks and moving head allover the disk platter is very expensive. But this is very inefficient for SSDs, as each SSD can handle multiple parallel writers, also this method is extremely inefficient for compaction as it causes write amplification and inefficient disk space usage. Our proposal is to have multiple active entrylogs and a configuration param on how many parallel entrylogs the system can have. This way one can have ability to configure to have less (may be one) ledger per entrylog. ### Comments from JIRA --- *Enrico Olivelli* 2017-04-20T07:28:28.619+0000 {quote} But this is very inefficient for HDDs {quote} [~jujjuri] did you mean SSD ? --- *Venkateswararao Jujjuri (JV)* 2017-04-21T16:42:29.203+0000 Yes I mean SSD. Corrected. Thanks. --- *Charan Reddy Guttapalem* 2017-05-18T15:01:22.566+0000 Issue: In Bookie's EntryLogger, we are having only one current active entryLog and all the ledger/entries go to the same entrylog. This is perfect for HDDs as file syncs, seeks and moving head allover the disk platter is very expensive. But having single active Entry Log is inefficient for SSDs, as each SSD can handle multiple parallel writers. Also, having single active EntryLog (irrespective of LedgerStorage type - interleaved/sorted), is inefficient for compaction, since entries of multiple ledgers will end up in an entrylog. Also in SortedLedgerStorage , in the addEntry request we flush EntryMemtable, if it reaches the sizelimit. Because of this we are

observing unpredictable tail latency for addEntry request. When EntryMemTable snapshot of size (64 MB) is flushed all at once, this may affect the journal addentry latency. Also, if the rate of new add requests surpasses the rate at which the EntryMemTable's previous snapshot is flushed, then at a point the current EntryMemTable map will reach the limit and since the previous snapshot flush is in progress, EntryMemTable will throttle new addRequests, which would affect addEntry latency. Goals: The main purpose of this feature is to have efficient Garbagecollection story by minimizing the amount of compactions required and the ability to reclaim the deleted ledger's space quicker. Also with this feature we can lay foreground for switching to InterleavedLedgerStorage from SortedLedgerStorage and get predictable tail latency. Proposal: So proposal here is to have multiple active entrylogs. Which will help with compaction performance and make SortedLedgerStorage redundant. Design Overview: - is to have server configuration specifying number of active entry logs per ledgerdir. - for backward compatibility (for existing behaviour) that config can be set to 0. - round-robin method will be used for choosing the active entry log for the current ledger in EntryLogger.addEntry method - if the total number of active entrylogs is more than or equal to number of active ledgers, then we get almost exclusivity - For implementing Round-Robin approach, we need to maintain state information of mapping of ledgerId to SlotId - there will be numberofledgerdirs*numberofactiveentrylogsperledgerdir slots. a slot is mapped to ledgerdir, but the activeentrylog of that slot will be rotated when it reaches the capacity. - By knowing the SlotId we can get the corresponding entryLogId associated to that slot. - If there is no entry for current ledger in the map, then we pick the next in order slot and add the mapping entry to the map. - Since Bookie won't be informed about the writeclose of the ledger, there is no easy way to know when to remove the mapping entry from the map. Considering it is just <long ledgerid, int slotid> mapentry, we may compromise on evicting policy. We can just use some Cache, which has eviction policy, timebased on last access - If a ledgerdir becomes full, then all the slots having entrylogs in that ledgerdir, should become inactive. The existing mappings, mappings of active ledgers to these slots (active entrylogs), should be updated to available active slots. - when ledgerdir becomes writable again, then the slots which were inactive should be made active and become eligible for round-robin distribution - For this feature I need to make changes to checkpoint logic. Currently with BOOKKEEPER-564 change, we are scheduling checkpoint only when current entrylog file is rotated. So we dont call 'flushCurrentLog' when we checkpoint. But for this feature, since there are going to be multiple active entrylogs, scheduling checkpoint when entrylog file is rotated, is not an option. So I need to call flushCurrentLogs when checkpoint is made for every 'flushinterval' period --- *Enrico Olivelli* 2017-05-19T06:49:35.104+0000 [~jujjuri] [~reddycharan18@gmail.com] This sound very interesting. Now I can see clearly way JV wrote on the mailing list that maybe clients could send some hint to the bookies that a ledger has been gracefully deleted/closed --- *Charan Reddy Guttapalem* 2017-06-02T00:22:59.045+0000 [~sijie@apache.org] created writeup for this work item and discussed about it in last community call (May 18th)

11. **title:** Multiple active entrylogs
    body: JIRA: https://issues.apache.org/jira/browse/BOOKKEEPER-1041 Reporter: Venkateswararao Jujjuri (JV) @jvrao Current bookkeeper is tuned for rotational HDDs. It has one active entrylog, and all the ledger/entries go to the same entrylog until it is rotated out. This is perfect for HDDs as seeks and moving head allover the disk platter is very expensive. But this is very inefficient for SSDs, as each SSD can handle multiple parallel writers, also this method is extremely inefficient for compaction as it causes write amplification and inefficient disk space usage. Our proposal is to have multiple active entrylogs and a configuration param on how many parallel entrylogs the system can have. This way one can have ability to configure to have less (may be one) ledger per entrylog. ### Comments from JIRA --- *Enrico Olivelli* 2017-04-20T07:28:28.619+0000 {quote} But this is very inefficient for HDDs {quote} [~jujjuri] did you mean SSD ? --- *Venkateswararao Jujjuri (JV)* 2017-04-21T16:42:29.203+0000 Yes I mean SSD. Corrected. Thanks. --- *Charan Reddy Guttapalem* 2017-05-18T15:01:22.566+0000 Issue: In Bookie's EntryLogger, we are having only one current active entryLog and all the ledger/entries go to the same entrylog. This is perfect for HDDs as file syncs, seeks and moving head allover the disk platter is very expensive. But having single active Entry Log is inefficient for SSDs, as each SSD can handle multiple parallel writers. Also, having single active EntryLog (irrespective of LedgerStorage type - interleaved/sorted), is inefficient for compaction, since entries of multiple ledgers will end up in an entrylog. Also in SortedLedgerStorage , in the addEntry request we flush EntryMemtable, if it reaches the sizelimit. Because of this we are observing unpredictable tail latency for addEntry request. When EntryMemTable snapshot of size (64 MB) is flushed all at once, this may affect the journal addentry latency. Also, if the rate of new add requests surpasses the rate at which the EntryMemTable's previous snapshot is flushed, then at a point the current EntryMemTable map will reach the limit and since the previous snapshot flush is in progress, EntryMemTable will throttle new addRequests, which would affect addEntry latency. Goals: The main purpose of this feature is to have efficient Garbagecollection story by minimizing the amount of compactions required and the ability to reclaim the deleted ledger's space quicker. Also with this feature we can lay foreground for switching to InterleavedLedgerStorage from SortedLedgerStorage and get predictable tail latency. Proposal: So proposal here is to have multiple active entrylogs. Which will help with compaction performance and make SortedLedgerStorage redundant. Design Overview: - is to have server configuration specifying number of active entry logs per ledgerdir. - for backward compatibility (for existing behaviour) that config can be set to 0. - round-robin method will be used for choosing the active entry log for the current ledger in EntryLogger.addEntry method - if the total number of active entrylogs is more than or equal to number of active ledgers, then we get almost exclusivity - For implementing Round-Robin approach, we need to maintain state information of mapping of ledgerId to SlotId - there will be numberofledgerdirs*numberofactiveentrylogsperledgerdir slots. a slot is mapped to ledgerdir, but the activeentrylog of that slot will be rotated when it reaches the capacity. - By knowing the SlotId we can get the corresponding entryLogId associated to that slot. - If there is no entry for current ledger in the map, then we pick the next in order slot and add the mapping entry to the map. - Since Bookie won't be informed about the writeclose of the ledger, there is no easy way to know when to remove the mapping entry from the map. Considering it is just <long ledgerid, int slotid> mapentry, we may compromise on evicting policy. We can just use some Cache, which has eviction policy, timebased on last access - If a ledgerdir becomes full, then all the slots having entrylogs in that ledgerdir, should become inactive. The existing mappings, mappings of active ledgers to these slots (active entrylogs), should be updated to available active slots. - when ledgerdir becomes writable again, then the slots which were inactive should be made active and become eligible for round-robin distribution - For this feature I need to make changes to checkpoint logic. Currently with BOOKKEEPER-564 change, we are scheduling checkpoint only when current entrylog file is rotated. So we dont call 'flushCurrentLog' when we checkpoint. But for this feature, since there are going to be multiple active entrylogs, scheduling checkpoint when entrylog file is rotated, is not an option. So I need to call flushCurrentLogs when checkpoint is made for every 'flushinterval' period --- *Enrico Olivelli* 2017-05-19T06:49:35.104+0000 [~jujjuri] [~reddycharan18@gmail.com] This sound very interesting. Now I can see clearly way JV wrote on the mailing list that maybe clients could send some hint to the bookies that a ledger has been gracefully deleted/closed --- *Charan Reddy Guttapalem* 2017-06-02T00:22:59.045+0000 [~sijie@apache.org] created writeup for this work item and discussed about it in last community call (May 18th)
    label: code-design
12. **title:** Multiple active entrylogs
    body: JIRA: https://issues.apache.org/jira/browse/BOOKKEEPER-1041 Reporter: Venkateswararao Jujjuri (JV) @jvrao Current bookkeeper is tuned for rotational HDDs. It has one active entrylog, and all the ledger/entries go to the same entrylog until it is rotated out. This is perfect for HDDs as seeks and moving head allover the disk platter is very expensive. But this is very inefficient for SSDs, as each SSD can handle multiple parallel writers, also this method is extremely inefficient for compaction as it causes write amplification and inefficient disk space usage. Our proposal is to have multiple active entrylogs and a configuration param on how many parallel entrylogs the system can have. This way one can have ability to configure to have less (may be one) ledger per entrylog. ### Comments from JIRA --- *Enrico Olivelli* 2017-04-20T07:28:28.619+0000 {quote} But this is very inefficient for HDDs {quote} [~jujjuri] did you mean SSD ? ---

Venkateswararao Jujjuri (JV) 2017-04-21T16:42:29.203+0000 Yes I mean SSD. Corrected. Thanks. --- *Charan Reddy Guttapalem* 2017-05-18T15:01:22.566+0000 Issue: In Bookie's EntryLogger, we are having only one current active entryLog and all the ledger/entries go to the same entrylog. This is perfect for HDDs as file syncs, seeks and moving head allover the disk platter is very expensive. But having single active Entry Log is inefficient for SSDs, as each SSD can handle multiple parallel writers. Also, having single active EntryLog (irrespective of LedgerStorage type - interleaved/sorted), is inefficient for compaction, since entries of multiple ledgers will end up in an entrylog. Also in SortedLedgerStorage , in the addEntry request we flush EntryMemtable, if it reaches the sizelimit. Because of this we are observing unpredictable tail latency for addEntry request. When EntryMemTable snapshot of size (64 MB) is flushed all at once, this may affect the journal addentry latency. Also, if the rate of new add requests surpasses the rate at which the EntryMemTable's previous snapshot is flushed, then at a point the current EntryMemTable map will reach the limit and since the previous snapshot flush is in progress, EntryMemTable will throttle new addRequests, which would affect addEntry latency. Goals: The main purpose of this feature is to have efficient Garbagecollection story by minimizing the amount of compactions required and the ability to reclaim the deleted ledger's space quicker. Also with this feature we can lay foreground for switching to InterleavedLedgerStorage from SortedLedgerStorage and get predictable tail latency. Proposal: So proposal here is to have multiple active entrylogs. Which will help with compaction performance and make SortedLedgerStorage redundant. Design Overview: - is to have server configuration specifying number of active entry logs per ledgerdir. - for backward compatibility (for existing behaviour) that config can be set to 0. - round-robin method will be used for choosing the active entry log for the current ledger in EntryLogger.addEntry method - if the total number of active entrylogs is more than or equal to number of active ledgers, then we get almost exclusivity - For implementing Round-Robin approach, we need to maintain state information of mapping of ledgerId to SlotId - there will be numberofledgerdirs*numberofactiveentrylogsperledgerdir slots. a slot is mapped to ledgerdir, but the activeentrylog of that slot will be rotated when it reaches the capacity. - By knowing the SlotId we can get the corresponding entryLogId associated to that slot. - If there is no entry for current ledger in the map, then we pick the next in order slot and add the mapping entry to the map. - Since Bookie won't be informed about the writeclose of the ledger, there is no easy way to know when to remove the mapping entry from the map. Considering it is just <long ledgerid, int slotid> mapentry, we may compromise on evicting policy. We can just use some Cache, which has eviction policy, timebased on last access - If a ledgerdir becomes full, then all the slots having entrylogs in that ledgerdir, should become inactive. The existing mappings, mappings of active ledgers to these slots (active entrylogs), should be updated to available active slots. - when ledgerdir becomes writable again, then the slots which were inactive should be made active and become eligible for round-robin distribution - For this feature I need to make changes to checkpoint logic. Currently with BOOKKEEPER-564 change, we are scheduling checkpoint only when current entrylog file is rotated. So we dont call 'flushCurrentLog' when we checkpoint. But for this feature, since there are going to be multiple active entrylogs, scheduling checkpoint when entrylog file is rotated, is not an option. So I need to call flushCurrentLogs when checkpoint is made for every 'flushinterval' period --- *Enrico Olivelli* 2017-05-19T06:49:35.104+0000 [~jujjuri] [~reddycharan18@gmail.com] This sound very interesting. Now I can see clearly way JV wrote on the mailing list that maybe clients could send some hint to the bookies that a ledger has been gracefully deleted/closed --- *Charan Reddy Guttapalem* 2017-06-02T00:22:59.045+0000 [~sijie@apache.org] created writeup for this work item and discussed about it in last community call (May 18th)

13. **title:** Multiple active entrylogs
body: JIRA: https://issues.apache.org/jira/browse/BOOKKEEPER-1041 Reporter: Venkateswararao Jujjuri (JV) @jvrao Current bookkeeper is tuned for rotational HDDs. It has one active entrylog, and all the ledger/entries go to the same entrylog until it is rotated out. This is perfect for HDDs as seeks and moving head allover the disk platter is very expensive. But this is very inefficient for SSDs, as each SSD can handle multiple parallel writers, also this method is extremely inefficient for compaction as it causes write amplification and inefficient disk space usage. Our proposal is to have multiple active entrylogs and a configuration param on how many parallel entrylogs the system can have. This way one can have ability to configure to have less (may be one) ledger per entrylog. ### Comments from JIRA --- *Enrico Olivelli* 2017-04-20T07:28:28.619+0000 {quote} But this is very inefficient for HDDs {quote} [~jujjuri] did you mean SSD ? --- *Venkateswararao Jujjuri (JV)* 2017-04-21T16:42:29.203+0000 Yes I mean SSD. Corrected. Thanks. --- *Charan Reddy Guttapalem* 2017-05-18T15:01:22.566+0000 Issue: In Bookie's EntryLogger, we are having only one current active entryLog and all the ledger/entries go to the same entrylog. This is perfect for HDDs as file syncs, seeks and moving head allover the disk platter is very expensive. But having single active Entry Log is inefficient for SSDs, as each SSD can handle multiple parallel writers. Also, having single active EntryLog (irrespective of LedgerStorage type - interleaved/sorted), is inefficient for compaction, since entries of multiple ledgers will end up in an entrylog. Also in SortedLedgerStorage , in the addEntry request we flush EntryMemtable, if it reaches the sizelimit. Because of this we are observing unpredictable tail latency for addEntry request. When EntryMemTable snapshot of size (64 MB) is flushed all at once, this may affect the journal addentry latency. Also, if the rate of new add requests surpasses the rate at which the EntryMemTable's previous snapshot is flushed, then at a point the current EntryMemTable map will reach the limit and since the previous snapshot flush is in progress, EntryMemTable will throttle new addRequests, which would affect addEntry latency. Goals: The main purpose of this feature is to have efficient Garbagecollection story by minimizing the amount of compactions required and the ability to reclaim the deleted ledger's space quicker. Also with this feature we can lay foreground for switching to InterleavedLedgerStorage from SortedLedgerStorage and get predictable tail latency. Proposal: So proposal here is to have multiple active entrylogs. Which will help with compaction performance and make SortedLedgerStorage redundant. Design Overview: - is to have server configuration specifying number of active entry logs per ledgerdir. - for backward compatibility (for existing behaviour) that config can be set to 0. - round-robin method will be used for choosing the active entry log for the current ledger in EntryLogger.addEntry method - if the total number of active entrylogs is more than or equal to number of active ledgers, then we get almost exclusivity - For implementing Round-Robin approach, we need to maintain state information of mapping of ledgerId to SlotId - there will be numberofledgerdirs*numberofactiveentrylogsperledgerdir slots. a slot is mapped to ledgerdir, but the activeentrylog of that slot will be rotated when it reaches the capacity. - By knowing the SlotId we can get the corresponding entryLogId associated to that slot. - If there is no entry for current ledger in the map, then we pick the next in order slot and add the mapping entry to the map. - Since Bookie won't be informed about the writeclose of the ledger, there is no easy way to know when to remove the mapping entry from the map. Considering it is just <long ledgerid, int slotid> mapentry, we may compromise on evicting policy. We can just use some Cache, which has eviction policy, timebased on last access - If a ledgerdir becomes full, then all the slots having entrylogs in that ledgerdir, should become inactive. The existing mappings, mappings of active ledgers to these slots (active entrylogs), should be updated to available active slots. - when ledgerdir becomes writable again, then the slots which were inactive should be made active and become eligible for round-robin distribution - For this feature I need to make changes to checkpoint logic. Currently with BOOKKEEPER-564 change, we are scheduling checkpoint only when current entrylog file is rotated. So we dont call 'flushCurrentLog' when we checkpoint. But for this feature, since there are going to be multiple active entrylogs, scheduling checkpoint when entrylog file is rotated, is not an option. So I need to call flushCurrentLogs when checkpoint is made for every 'flushinterval' period --- *Enrico Olivelli* 2017-05-19T06:49:35.104+0000 [~jujjuri] [~reddycharan18@gmail.com] This sound very interesting. Now I can see clearly way JV wrote on the mailing list that maybe clients could send some hint to the bookies that a ledger has been gracefully deleted/closed --- *Charan Reddy Guttapalem* 2017-06-02T00:22:59.045+0000 [~sijie@apache.org] created writeup for this work item and discussed about it in last community call (May 18th)
14. **title:** Multiple active entrylogs
body: JIRA: https://issues.apache.org/jira/browse/BOOKKEEPER-1041 Reporter: Venkateswararao Jujjuri (JV) @jvrao Current bookkeeper is tuned for rotational HDDs. It has one active entrylog, and all the ledger/entries go to the same entrylog until it is rotated out. This is

perfect for HDDs as seeks and moving head allover the disk platter is very expensive. But this is very inefficient for SSDs, as each SSD can handle multiple parallel writers, also this method is extremely inefficient for compaction as it causes write amplification and inefficient disk space usage. Our proposal is to have multiple active entrylogs and a configuration param on how many parallel entrylogs the system can have. This way one can have ability to configure to have less (may be one) ledger per entrylog. ### Comments from JIRA --- *Enrico Olivelli* 2017-04-20T07:28:28.619+0000 {quote} But this is very inefficient for HDDs {quote} [~jujjuri] did you mean SSD ? --- *Venkateswararao Jujjuri (JV)* 2017-04-21T16:42:29.203+0000 Yes I mean SSD. Corrected. Thanks. --- *Charan Reddy Guttapalem* 2017-05-18T15:01:22.566+0000 Issue: In Bookie's EntryLogger, we are having only one current active entryLog and all the ledger/entries go to the same entrylog. This is perfect for HDDs as file syncs, seeks and moving head allover the disk platter is very expensive. But having single active Entry Log is inefficient for SSDs, as each SSD can handle multiple parallel writers. Also, having single active EntryLog (irrespective of LedgerStorage type - interleaved/sorted), is inefficient for compaction, since entries of multiple ledgers will end up in an entrylog. Also in SortedLedgerStorage , in the addEntry request we flush EntryMemtable, if it reaches the sizelimit. Because of this we are observing unpredictable tail latency for addEntry request. When EntryMemTable snapshot of size (64 MB) is flushed all at once, this may affect the journal addentry latency. Also, if the rate of new add requests surpasses the rate at which the EntryMemTable's previous snapshot is flushed, then at a point the current EntryMemTable map will reach the limit and since the previous snapshot flush is in progress, EntryMemTable will throttle new addRequests, which would affect addEntry latency. Goals: The main purpose of this feature is to have efficient Garbagecollection story by minimizing the amount of compactions required and the ability to reclaim the deleted ledger's space quicker. Also with this feature we can lay foreground for switching to InterleavedLedgerStorage from SortedLedgerStorage and get predictable tail latency. Proposal: So proposal here is to have multiple active entrylogs. Which will help with compaction performance and make SortedLedgerStorage redundant. Design Overview: - is to have server configuration specifying number of active entry logs per ledgerdir. - for backward compatibility (for existing behaviour) that config can be set to 0. - round-robin method will be used for choosing the active entry log for the current ledger in EntryLogger.addEntry method - if the total number of active entrylogs is more than or equal to number of active ledgers, then we get almost exclusivity - For implementing Round-Robin approach, we need to maintain state information of mapping of ledgerId to SlotId - there will be numberofledgerdirs*numberofactiveentrylogsperledgerdir slots. a slot is mapped to ledgerdir, but the activeentrylog of that slot will be rotated when it reaches the capacity. - By knowing the SlotId we can get the corresponding entryLogId associated to that slot. - If there is no entry for current ledger in the map, then we pick the next in order slot and add the mapping entry to the map. - Since Bookie won't be informed about the writeclose of the ledger, there is no easy way to know when to remove the mapping entry from the map. Considering it is just <long ledgerid, int slotid> mapentry, we may compromise on evicting policy. We can just use some Cache, which has eviction policy, timebased on last access - If a ledgerdir becomes full, then all the slots having entrylogs in that ledgerdir, should become inactive. The existing mappings, mappings of active ledgers to these slots (active entrylogs), should be updated to available active slots. - when ledgerdir becomes writable again, then the slots which were inactive should be made active and become eligible for round-robin distribution - For this feature I need to make changes to checkpoint logic. Currently with BOOKKEEPER-564 change, we are scheduling checkpoint only when current entrylog file is rotated. So we dont call 'flushCurrentLog' when we checkpoint. But for this feature, since there are going to be multiple active entrylogs, scheduling checkpoint when entrylog file is rotated, is not an option. So I need to call flushCurrentLogs when checkpoint is made for every 'flushinterval' period --- *Enrico Olivelli* 2017-05-19T06:49:35.104+0000 [~jujjuri] [~reddycharan18@gmail.com] This sound very interesting. Now I can see clearly way JV wrote on the mailing list that maybe clients could send some hint to the bookies that a ledger has been gracefully deleted/closed --- *Charan Reddy Guttapalem* 2017-06-02T00:22:59.045+0000 [~sijie@apache.org] created writeup for this work item and discussed about it in last community call (May 18th)

15. **title:** Multiple active entrylogs
body: JIRA: https://issues.apache.org/jira/browse/BOOKKEEPER-1041 Reporter: Venkateswararao Jujjuri (JV) @jvrao Current bookkeeper is tuned for rotational HDDs. It has one active entrylog, and all the ledger/entries go to the same entrylog until it is rotated out. This is perfect for HDDs as seeks and moving head allover the disk platter is very expensive. But this is very inefficient for SSDs, as each SSD can handle multiple parallel writers, also this method is extremely inefficient for compaction as it causes write amplification and inefficient disk space usage. Our proposal is to have multiple active entrylogs and a configuration param on how many parallel entrylogs the system can have. This way one can have ability to configure to have less (may be one) ledger per entrylog. ### Comments from JIRA --- *Enrico Olivelli* 2017-04-20T07:28:28.619+0000 {quote} But this is very inefficient for HDDs {quote} [~jujjuri] did you mean SSD ? --- *Venkateswararao Jujjuri (JV)* 2017-04-21T16:42:29.203+0000 Yes I mean SSD. Corrected. Thanks. --- *Charan Reddy Guttapalem* 2017-05-18T15:01:22.566+0000 Issue: In Bookie's EntryLogger, we are having only one current active entryLog and all the ledger/entries go to the same entrylog. This is perfect for HDDs as file syncs, seeks and moving head allover the disk platter is very expensive. But having single active Entry Log is inefficient for SSDs, as each SSD can handle multiple parallel writers. Also, having single active EntryLog (irrespective of LedgerStorage type - interleaved/sorted), is inefficient for compaction, since entries of multiple ledgers will end up in an entrylog. Also in SortedLedgerStorage , in the addEntry request we flush EntryMemtable, if it reaches the sizelimit. Because of this we are observing unpredictable tail latency for addEntry request. When EntryMemTable snapshot of size (64 MB) is flushed all at once, this may affect the journal addentry latency. Also, if the rate of new add requests surpasses the rate at which the EntryMemTable's previous snapshot is flushed, then at a point the current EntryMemTable map will reach the limit and since the previous snapshot flush is in progress, EntryMemTable will throttle new addRequests, which would affect addEntry latency. Goals: The main purpose of this feature is to have efficient Garbagecollection story by minimizing the amount of compactions required and the ability to reclaim the deleted ledger's space quicker. Also with this feature we can lay foreground for switching to InterleavedLedgerStorage from SortedLedgerStorage and get predictable tail latency. Proposal: So proposal here is to have multiple active entrylogs. Which will help with compaction performance and make SortedLedgerStorage redundant. Design Overview: - is to have server configuration specifying number of active entry logs per ledgerdir. - for backward compatibility (for existing behaviour) that config can be set to 0. - round-robin method will be used for choosing the active entry log for the current ledger in EntryLogger.addEntry method - if the total number of active entrylogs is more than or equal to number of active ledgers, then we get almost exclusivity - For implementing Round-Robin approach, we need to maintain state information of mapping of ledgerId to SlotId - there will be numberofledgerdirs*numberofactiveentrylogsperledgerdir slots. a slot is mapped to ledgerdir, but the activeentrylog of that slot will be rotated when it reaches the capacity. - By knowing the SlotId we can get the corresponding entryLogId associated to that slot. - If there is no entry for current ledger in the map, then we pick the next in order slot and add the mapping entry to the map. - Since Bookie won't be informed about the writeclose of the ledger, there is no easy way to know when to remove the mapping entry from the map. Considering it is just <long ledgerid, int slotid> mapentry, we may compromise on evicting policy. We can just use some Cache, which has eviction policy, timebased on last access - If a ledgerdir becomes full, then all the slots having entrylogs in that ledgerdir, should become inactive. The existing mappings, mappings of active ledgers to these slots (active entrylogs), should be updated to available active slots. - when ledgerdir becomes writable again, then the slots which were inactive should be made active and become eligible for round-robin distribution - For this feature I need to make changes to checkpoint logic. Currently with BOOKKEEPER-564 change, we are scheduling checkpoint only when current entrylog file is rotated. So we dont call 'flushCurrentLog' when we checkpoint. But for this feature, since there are going to be multiple active entrylogs, scheduling checkpoint when entrylog file is rotated, is not an option. So I need to call flushCurrentLogs when checkpoint is made for every 'flushinterval' period --- *Enrico Olivelli* 2017-05-19T06:49:35.104+0000 [~jujjuri] [~reddycharan18@gmail.com] This sound very interesting. Now I can see clearly way JV wrote on the mailing list that maybe clients could send some hint to the bookies that a ledger has been gracefully deleted/closed --- *Charan Reddy

Guttapalem* 2017-06-02T00:22:59.045+0000 [~sijie@apache.org] created writeup for this work item and discussed about it in last community call (May 18th)

16. **title:** Multiple active entrylogs
body: JIRA: https://issues.apache.org/jira/browse/BOOKKEEPER-1041 Reporter: Venkateswararao Jujjuri (JV) @jvrao Current bookkeeper is tuned for rotational HDDs. It has one active entrylog, and all the ledger/entries go to the same entrylog until it is rotated out. This is perfect for HDDs as seeks and moving head allover the disk platter is very expensive. But this is very inefficient for SSDs, as each SSD can handle multiple parallel writers, also this method is extremely inefficient for compaction as it causes write amplification and inefficient disk space usage. Our proposal is to have multiple active entrylogs and a configuration param on how many parallel entrylogs the system can have. This way one can have ability to configure to have less (may be one) ledger per entrylog. ### Comments from JIRA --- *Enrico Olivelli* 2017-04-20T07:28:28.619+0000 {quote} But this is very inefficient for HDDs {quote} [~jujjuri] did you mean SSD ? --- *Venkateswararao Jujjuri (JV)* 2017-04-21T16:42:29.203+0000 Yes I mean SSD. Corrected. Thanks. --- *Charan Reddy Guttapalem* 2017-05-18T15:01:22.566+0000 Issue: In Bookie's EntryLogger, we are having only one current active entryLog and all the ledger/entries go to the same entrylog. This is perfect for HDDs as file syncs, seeks and moving head allover the disk platter is very expensive. But having single active Entry Log is inefficient for SSDs, as each SSD can handle multiple parallel writers. Also, having single active EntryLog (irrespective of LedgerStorage type - interleaved/sorted), is inefficient for compaction, since entries of multiple ledgers will end up in an entrylog. Also in SortedLedgerStorage , in the addEntry request we flush EntryMemtable, if it reaches the sizelimit. Because of this we are observing unpredictable tail latency for addEntry request. When EntryMemTable snapshot of size (64 MB) is flushed all at once, this may affect the journal addentry latency. Also, if the rate of new add requests surpasses the rate at which the EntryMemTable's previous snapshot is flushed, then at a point the current EntryMemTable map will reach the limit and since the previous snapshot flush is in progress, EntryMemTable will throttle new addRequests, which would affect addEntry latency. Goals: The main purpose of this feature is to have efficient Garbagecollection story by minimizing the amount of compactions required and the ability to reclaim the deleted ledger's space quicker. Also with this feature we can lay foreground for switching to InterleavedLedgerStorage from SortedLedgerStorage and get predictable tail latency. Proposal: So proposal here is to have multiple active entrylogs. Which will help with compaction performance and make SortedLedgerStorage redundant. Design Overview: - is to have server configuration specifying number of active entry logs per ledgerdir. - for backward compatibility (for existing behaviour) that config can be set to 0. - round-robin method will be used for choosing the active entry log for the current ledger in EntryLogger.addEntry method - if the total number of active entrylogs is more than or equal to number of active ledgers, then we get almost exclusivity - For implementing Round-Robin approach, we need to maintain state information of mapping of ledgerId to SlotId - there will be numberofledgerdirs*numberofactiveentrylogsperledgerdir slots. a slot is mapped to ledgerdir, but the activeentrylog of that slot will be rotated when it reaches the capacity. - By knowing the SlotId we can get the corresponding entryLogId associated to that slot. - If there is no entry for current ledger in the map, then we pick the next in order slot and add the mapping entry to the map. - Since Bookie won't be informed about the writeclose of the ledger, there is no easy way to know when to remove the mapping entry from the map. Considering it is just <long ledgerid, int slotid> mapentry, we may compromise on evicting policy. We can just use some Cache, which has eviction policy, timebased on last access - If a ledgerdir becomes full, then all the slots having entrylogs in that ledgerdir, should become inactive. The existing mappings, mappings of active ledgers to these slots (active entrylogs), should be updated to available active slots. - when ledgerdir becomes writable again, then the slots which were inactive should be made active and become eligible for round-robin distribution - For this feature I need to make changes to checkpoint logic. Currently with BOOKKEEPER-564 change, we are scheduling checkpoint only when current entrylog file is rotated. So we dont call 'flushCurrentLog' when we checkpoint. But for this feature, since there are going to be multiple active entrylogs, scheduling checkpoint when entrylog file is rotated, is not an option. So I need to call flushCurrentLogs when checkpoint is made for every 'flushinterval' period --- *Enrico Olivelli* 2017-05-19T06:49:35.104+0000 [~jujjuri] [~reddycharan18@gmail.com] This sound very interesting. Now I can see clearly way JV wrote on the mailing list that maybe clients could send some hint to the bookies that a ledger has been gracefully deleted/closed --- *Charan Reddy Guttapalem* 2017-06-02T00:22:59.045+0000 [~sijie@apache.org] created writeup for this work item and discussed about it in last community call (May 18th)

17. **title:** Multiple active entrylogs
body: JIRA: https://issues.apache.org/jira/browse/BOOKKEEPER-1041 Reporter: Venkateswararao Jujjuri (JV) @jvrao Current bookkeeper is tuned for rotational HDDs. It has one active entrylog, and all the ledger/entries go to the same entrylog until it is rotated out. This is perfect for HDDs as seeks and moving head allover the disk platter is very expensive. But this is very inefficient for SSDs, as each SSD can handle multiple parallel writers, also this method is extremely inefficient for compaction as it causes write amplification and inefficient disk space usage. Our proposal is to have multiple active entrylogs and a configuration param on how many parallel entrylogs the system can have. This way one can have ability to configure to have less (may be one) ledger per entrylog. ### Comments from JIRA --- *Enrico Olivelli* 2017-04-20T07:28:28.619+0000 {quote} But this is very inefficient for HDDs {quote} [~jujjuri] did you mean SSD ? --- *Venkateswararao Jujjuri (JV)* 2017-04-21T16:42:29.203+0000 Yes I mean SSD. Corrected. Thanks. --- *Charan Reddy Guttapalem* 2017-05-18T15:01:22.566+0000 Issue: In Bookie's EntryLogger, we are having only one current active entryLog and all the ledger/entries go to the same entrylog. This is perfect for HDDs as file syncs, seeks and moving head allover the disk platter is very expensive. But having single active Entry Log is inefficient for SSDs, as each SSD can handle multiple parallel writers. Also, having single active EntryLog (irrespective of LedgerStorage type - interleaved/sorted), is inefficient for compaction, since entries of multiple ledgers will end up in an entrylog. Also in SortedLedgerStorage , in the addEntry request we flush EntryMemtable, if it reaches the sizelimit. Because of this we are observing unpredictable tail latency for addEntry request. When EntryMemTable snapshot of size (64 MB) is flushed all at once, this may affect the journal addentry latency. Also, if the rate of new add requests surpasses the rate at which the EntryMemTable's previous snapshot is flushed, then at a point the current EntryMemTable map will reach the limit and since the previous snapshot flush is in progress, EntryMemTable will throttle new addRequests, which would affect addEntry latency. Goals: The main purpose of this feature is to have efficient Garbagecollection story by minimizing the amount of compactions required and the ability to reclaim the deleted ledger's space quicker. Also with this feature we can lay foreground for switching to InterleavedLedgerStorage from SortedLedgerStorage and get predictable tail latency. Proposal: So proposal here is to have multiple active entrylogs. Which will help with compaction performance and make SortedLedgerStorage redundant. Design Overview: - is to have server configuration specifying number of active entry logs per ledgerdir. - for backward compatibility (for existing behaviour) that config can be set to 0. - round-robin method will be used for choosing the active entry log for the current ledger in EntryLogger.addEntry method - if the total number of active entrylogs is more than or equal to number of active ledgers, then we get almost exclusivity - For implementing Round-Robin approach, we need to maintain state information of mapping of ledgerId to SlotId - there will be numberofledgerdirs*numberofactiveentrylogsperledgerdir slots. a slot is mapped to ledgerdir, but the activeentrylog of that slot will be rotated when it reaches the capacity. - By knowing the SlotId we can get the corresponding entryLogId associated to that slot. - If there is no entry for current ledger in the map, then we pick the next in order slot and add the mapping entry to the map. - Since Bookie won't be informed about the writeclose of the ledger, there is no easy way to know when to remove the mapping entry from the map. Considering it is just <long ledgerid, int slotid> mapentry, we may compromise on evicting policy. We can just use some Cache, which has eviction policy, timebased on last access - If a ledgerdir becomes full, then all the slots having entrylogs in that ledgerdir, should become inactive. The existing mappings, mappings of active ledgers to these slots (active entrylogs), should be updated to available active slots. - when ledgerdir becomes writable again, then the slots which were inactive should be made active and become eligible for round-robin distribution - For this feature I need to make changes to checkpoint logic. Currently with BOOKKEEPER-564

change, we are scheduling checkpoint only when current entrylog file is rotated. So we dont call 'flushCurrentLog' when we checkpoint. But for this feature, since there are going to be multiple active entrylogs, scheduling checkpoint when entrylog file is rotated, is not an option. So I need to call flushCurrentLogs when checkpoint is made for every 'flushinterval' period --- *Enrico Olivelli* 2017-05-19T06:49:35.104+0000 [~jujjuri] [~reddycharan18@gmail.com] This sound very interesting. Now I can see clearly way JV wrote on the mailing list that maybe clients could send some hint to the bookies that a ledger has been gracefully deleted/closed --- *Charan Reddy Guttapalem* 2017-06-02T00:22:59.045+0000 [~sijie@apache.org] created writeup for this work item and discussed about it in last community call (May 18th)
label: code-design

18. **title:** Multiple active entrylogs
body: JIRA: https://issues.apache.org/jira/browse/BOOKKEEPER-1041 Reporter: Venkateswararao Jujjuri (JV) @jvrao Current bookkeeper is tuned for rotational HDDs. It has one active entrylog, and all the ledger/entries go to the same entrylog until it is rotated out. This is perfect for HDDs as seeks and moving head allover the disk platter is very expensive. But this is very inefficient for SSDs, as each SSD can handle multiple parallel writers, also this method is extremely inefficient for compaction as it causes write amplification and inefficient disk space usage. Our proposal is to have multiple active entrylogs and a configuration param on how many parallel entrylogs the system can have. This way one can have ability to configure to have less (may be one) ledger per entrylog. ### Comments from JIRA --- *Enrico Olivelli* 2017-04-20T07:28:28.619+0000 {quote} But this is very inefficient for HDDs {quote} [~jujjuri] did you mean SSD ? --- *Venkateswararao Jujjuri (JV)* 2017-04-21T16:42:29.203+0000 Yes I mean SSD. Corrected. Thanks. --- *Charan Reddy Guttapalem* 2017-05-18T15:01:22.566+0000 Issue: In Bookie's EntryLogger, we are having only one current active entryLog and all the ledger/entries go to the same entrylog. This is perfect for HDDs as file syncs, seeks and moving head allover the disk platter is very expensive. But having single active Entry Log is inefficient for SSDs, as each SSD can handle multiple parallel writers. Also, having single active EntryLog (irrespective of LedgerStorage type - interleaved/sorted), is inefficient for compaction, since entries of multiple ledgers will end up in an entrylog. Also in SortedLedgerStorage , in the addEntry request we flush EntryMemtable, if it reaches the sizelimit. Because of this we are observing unpredictable tail latency for addEntry request. When EntryMemTable snapshot of size (64 MB) is flushed all at once, this may affect the journal addentry latency. Also, if the rate of new add requests surpasses the rate at which the EntryMemTable's previous snapshot is flushed, then at a point the current EntryMemTable map will reach the limit and since the previous snapshot flush is in progress, EntryMemTable will throttle new addRequests, which would affect addEntry latency. Goals: The main purpose of this feature is to have efficient Garbagecollection story by minimizing the amount of compactions required and the ability to reclaim the deleted ledger's space quicker. Also with this feature we can lay foreground for switching to InterleavedLedgerStorage from SortedLedgerStorage and get predictable tail latency. Proposal: So proposal here is to have multiple active entrylogs. Which will help with compaction performance and make SortedLedgerStorage redundant. Design Overview: - is to have server configuration specifying number of active entry logs per ledgerdir. - for backward compatibility (for existing behaviour) that config can be set to 0. - round-robin method will be used for choosing the active entry log for the current ledger in EntryLogger.addEntry method - if the total number of active entrylogs is more than or equal to number of active ledgers, then we get almost exclusivity - For implementing Round-Robin approach, we need to maintain state information of mapping of ledgerId to SlotId - there will be numberofledgerdirs*numberofactiveentrylogsperledgerdir slots. a slot is mapped to ledgerdir, but the activeentrylog of that slot will be rotated when it reaches the capacity. - By knowing the SlotId we can get the corresponding entryLogId associated to that slot. - If there is no entry for current ledger in the map, then we pick the next in order slot and add the mapping entry to the map. - Since Bookie won't be informed about the writeclose of the ledger, there is no easy way to know when to remove the mapping entry from the map. Considering it is just <long ledgerid, int slotid> mapentry, we may compromise on evicting policy. We can just use some Cache, which has eviction policy, timebased on last access - If a ledgerdir becomes full, then all the slots having entrylogs in that ledgerdir, should become inactive. The existing mappings, mappings of active ledgers to these slots (active entrylogs), should be updated to available active slots. - when ledgerdir becomes writable again, then the slots which were inactive should be made active and become eligible for round-robin distribution - For this feature I need to make changes to checkpoint logic. Currently with BOOKKEEPER-564 change, we are scheduling checkpoint only when current entrylog file is rotated. So we dont call 'flushCurrentLog' when we checkpoint. But for this feature, since there are going to be multiple active entrylogs, scheduling checkpoint when entrylog file is rotated, is not an option. So I need to call flushCurrentLogs when checkpoint is made for every 'flushinterval' period --- *Enrico Olivelli* 2017-05-19T06:49:35.104+0000 [~jujjuri] [~reddycharan18@gmail.com] This sound very interesting. Now I can see clearly way JV wrote on the mailing list that maybe clients could send some hint to the bookies that a ledger has been gracefully deleted/closed --- *Charan Reddy Guttapalem* 2017-06-02T00:22:59.045+0000 [~sijie@apache.org] created writeup for this work item and discussed about it in last community call (May 18th)
label: code-design

19. **title:** Multiple active entrylogs
body: JIRA: https://issues.apache.org/jira/browse/BOOKKEEPER-1041 Reporter: Venkateswararao Jujjuri (JV) @jvrao Current bookkeeper is tuned for rotational HDDs. It has one active entrylog, and all the ledger/entries go to the same entrylog until it is rotated out. This is perfect for HDDs as seeks and moving head allover the disk platter is very expensive. But this is very inefficient for SSDs, as each SSD can handle multiple parallel writers, also this method is extremely inefficient for compaction as it causes write amplification and inefficient disk space usage. Our proposal is to have multiple active entrylogs and a configuration param on how many parallel entrylogs the system can have. This way one can have ability to configure to have less (may be one) ledger per entrylog. ### Comments from JIRA --- *Enrico Olivelli* 2017-04-20T07:28:28.619+0000 {quote} But this is very inefficient for HDDs {quote} [~jujjuri] did you mean SSD ? --- *Venkateswararao Jujjuri (JV)* 2017-04-21T16:42:29.203+0000 Yes I mean SSD. Corrected. Thanks. --- *Charan Reddy Guttapalem* 2017-05-18T15:01:22.566+0000 Issue: In Bookie's EntryLogger, we are having only one current active entryLog and all the ledger/entries go to the same entrylog. This is perfect for HDDs as file syncs, seeks and moving head allover the disk platter is very expensive. But having single active Entry Log is inefficient for SSDs, as each SSD can handle multiple parallel writers. Also, having single active EntryLog (irrespective of LedgerStorage type - interleaved/sorted), is inefficient for compaction, since entries of multiple ledgers will end up in an entrylog. Also in SortedLedgerStorage , in the addEntry request we flush EntryMemtable, if it reaches the sizelimit. Because of this we are observing unpredictable tail latency for addEntry request. When EntryMemTable snapshot of size (64 MB) is flushed all at once, this may affect the journal addentry latency. Also, if the rate of new add requests surpasses the rate at which the EntryMemTable's previous snapshot is flushed, then at a point the current EntryMemTable map will reach the limit and since the previous snapshot flush is in progress, EntryMemTable will throttle new addRequests, which would affect addEntry latency. Goals: The main purpose of this feature is to have efficient Garbagecollection story by minimizing the amount of compactions required and the ability to reclaim the deleted ledger's space quicker. Also with this feature we can lay foreground for switching to InterleavedLedgerStorage from SortedLedgerStorage and get predictable tail latency. Proposal: So proposal here is to have multiple active entrylogs. Which will help with compaction performance and make SortedLedgerStorage redundant. Design Overview: - is to have server configuration specifying number of active entry logs per ledgerdir. - for backward compatibility (for existing behaviour) that config can be set to 0. - round-robin method will be used for choosing the active entry log for the current ledger in EntryLogger.addEntry method - if the total number of active entrylogs is more than or equal to number of active ledgers, then we get almost exclusivity - For implementing Round-Robin approach, we need to maintain state information of mapping of ledgerId to SlotId - there will be numberofledgerdirs*numberofactiveentrylogsperledgerdir slots. a slot is mapped to ledgerdir, but the activeentrylog of that slot will be rotated when it reaches the capacity. - By knowing the SlotId we can get the corresponding

entryLogId associated to that slot. - If there is no entry for current ledger in the map, then we pick the next in order slot and add the mapping entry to the map. - Since Bookie won't be informed about the writeclose of the ledger, there is no easy way to know when to remove the mapping entry from the map. Considering it is just <long ledgerid, int slotid> mapentry, we may compromise on evicting policy. We can just use some Cache, which has eviction policy, timebased on last access - If a ledgerdir becomes full, then all the slots having entrylogs in that ledgerdir, should become inactive. The existing mappings, mappings of active ledgers to these slots (active entrylogs), should be updated to available active slots. - when ledgerdir becomes writable again, then the slots which were inactive should be made active and become eligible for round-robin distribution - For this feature I need to make changes to checkpoint logic. Currently with BOOKKEEPER-564 change, we are scheduling checkpoint only when current entrylog file is rotated. So we dont call 'flushCurrentLog' when we checkpoint. But for this feature, since there are going to be multiple active entrylogs, scheduling checkpoint when entrylog file is rotated, is not an option. So I need to call flushCurrentLogs when checkpoint is made for every 'flushinterval' period --- *Enrico Olivelli* 2017-05-19T06:49:35.104+0000 [~jujjuri] [~reddycharan18@gmail.com] This sound very interesting. Now I can see clearly way JV wrote on the mailing list that maybe clients could send some hint to the bookies that a ledger has been gracefully deleted/closed --- *Charan Reddy Guttapalem* 2017-06-02T00:22:59.045+0000 [~sijie@apache.org] created writeup for this work item and discussed about it in last community call (May 18th)
label: code-design

20. **title:** Multiple active entrylogs
body: JIRA: https://issues.apache.org/jira/browse/BOOKKEEPER-1041 Reporter: Venkateswararao Jujjuri (JV) @jvrao Current bookkeeper is tuned for rotational HDDs. It has one active entrylog, and all the ledger/entries go to the same entrylog until it is rotated out. This is perfect for HDDs as seeks and moving head allover the disk platter is very expensive. But this is very inefficient for SSDs, as each SSD can handle multiple parallel writers, also this method is extremely inefficient for compaction as it causes write amplification and inefficient disk space usage. Our proposal is to have multiple active entrylogs and a configuration param on how many parallel entrylogs the system can have. This way one can have ability to configure to have less (may be one) ledger per entrylog. ### Comments from JIRA --- *Enrico Olivelli* 2017-04-20T07:28:28.619+0000 {quote} But this is very inefficient for HDDs {quote} [~jujjuri] did you mean SSD ? --- *Venkateswararao Jujjuri (JV)* 2017-04-21T16:42:29.203+0000 Yes I mean SSD. Corrected. Thanks. --- *Charan Reddy Guttapalem* 2017-05-18T15:01:22.566+0000 Issue: In Bookie's EntryLogger, we are having only one current active entryLog and all the ledger/entries go to the same entrylog. This is perfect for HDDs as file syncs, seeks and moving head allover the disk platter is very expensive. But having single active Entry Log is inefficient for SSDs, as each SSD can handle multiple parallel writers. Also, having single active EntryLog (irrespective of LedgerStorage type - interleaved/sorted), is inefficient for compaction, since entries of multiple ledgers will end up in an entrylog. Also in SortedLedgerStorage , in the addEntry request we flush EntryMemtable, if it reaches the sizelimit. Because of this we are observing unpredictable tail latency for addEntry request. When EntryMemTable snapshot of size (64 MB) is flushed all at once, this may affect the journal addentry latency. Also, if the rate of new add requests surpasses the rate at which the EntryMemTable's previous snapshot is flushed, then at a point the current EntryMemTable map will reach the limit and since the previous snapshot flush is in progress, EntryMemTable will throttle new addRequests, which would affect addEntry latency. Goals: The main purpose of this feature is to have efficient Garbagecollection story by minimizing the amount of compactions required and the ability to reclaim the deleted ledger's space quicker. Also with this feature we can lay foreground for switching to InterleavedLedgerStorage from SortedLedgerStorage and get predictable tail latency. Proposal: So proposal here is to have multiple active entrylogs. Which will help with compaction performance and make SortedLedgerStorage redundant. Design Overview: - is to have server configuration specifying number of active entry logs per ledgerdir. - for backward compatibility (for existing behaviour) that config can be set to 0. - round-robin method will be used for choosing the active entry log for the current ledger in EntryLogger.addEntry method - if the total number of active entrylogs is more than or equal to number of active ledgers, then we get almost exclusivity - For implementing Round-Robin approach, we need to maintain state information of mapping of ledgerId to SlotId - there will be numberofledgerdirs*numberofactiveentrylogsperledgerdir slots. a slot is mapped to ledgerdir, but the activeentrylog of that slot will be rotated when it reaches the capacity. - By knowing the SlotId we can get the corresponding entryLogId associated to that slot. - If there is no entry for current ledger in the map, then we pick the next in order slot and add the mapping entry to the map. - Since Bookie won't be informed about the writeclose of the ledger, there is no easy way to know when to remove the mapping entry from the map. Considering it is just <long ledgerid, int slotid> mapentry, we may compromise on evicting policy. We can just use some Cache, which has eviction policy, timebased on last access - If a ledgerdir becomes full, then all the slots having entrylogs in that ledgerdir, should become inactive. The existing mappings, mappings of active ledgers to these slots (active entrylogs), should be updated to available active slots. - when ledgerdir becomes writable again, then the slots which were inactive should be made active and become eligible for round-robin distribution - For this feature I need to make changes to checkpoint logic. Currently with BOOKKEEPER-564 change, we are scheduling checkpoint only when current entrylog file is rotated. So we dont call 'flushCurrentLog' when we checkpoint. But for this feature, since there are going to be multiple active entrylogs, scheduling checkpoint when entrylog file is rotated, is not an option. So I need to call flushCurrentLogs when checkpoint is made for every 'flushinterval' period --- *Enrico Olivelli* 2017-05-19T06:49:35.104+0000 [~jujjuri] [~reddycharan18@gmail.com] This sound very interesting. Now I can see clearly way JV wrote on the mailing list that maybe clients could send some hint to the bookies that a ledger has been gracefully deleted/closed --- *Charan Reddy Guttapalem* 2017-06-02T00:22:59.045+0000 [~sijie@apache.org] created writeup for this work item and discussed about it in last community call (May 18th)

21. **title:** Multiple active entrylogs
body: JIRA: https://issues.apache.org/jira/browse/BOOKKEEPER-1041 Reporter: Venkateswararao Jujjuri (JV) @jvrao Current bookkeeper is tuned for rotational HDDs. It has one active entrylog, and all the ledger/entries go to the same entrylog until it is rotated out. This is perfect for HDDs as seeks and moving head allover the disk platter is very expensive. But this is very inefficient for SSDs, as each SSD can handle multiple parallel writers, also this method is extremely inefficient for compaction as it causes write amplification and inefficient disk space usage. Our proposal is to have multiple active entrylogs and a configuration param on how many parallel entrylogs the system can have. This way one can have ability to configure to have less (may be one) ledger per entrylog. ### Comments from JIRA --- *Enrico Olivelli* 2017-04-20T07:28:28.619+0000 {quote} But this is very inefficient for HDDs {quote} [~jujjuri] did you mean SSD ? --- *Venkateswararao Jujjuri (JV)* 2017-04-21T16:42:29.203+0000 Yes I mean SSD. Corrected. Thanks. --- *Charan Reddy Guttapalem* 2017-05-18T15:01:22.566+0000 Issue: In Bookie's EntryLogger, we are having only one current active entryLog and all the ledger/entries go to the same entrylog. This is perfect for HDDs as file syncs, seeks and moving head allover the disk platter is very expensive. But having single active Entry Log is inefficient for SSDs, as each SSD can handle multiple parallel writers. Also, having single active EntryLog (irrespective of LedgerStorage type - interleaved/sorted), is inefficient for compaction, since entries of multiple ledgers will end up in an entrylog. Also in SortedLedgerStorage , in the addEntry request we flush EntryMemtable, if it reaches the sizelimit. Because of this we are observing unpredictable tail latency for addEntry request. When EntryMemTable snapshot of size (64 MB) is flushed all at once, this may affect the journal addentry latency. Also, if the rate of new add requests surpasses the rate at which the EntryMemTable's previous snapshot is flushed, then at a point the current EntryMemTable map will reach the limit and since the previous snapshot flush is in progress, EntryMemTable will throttle new addRequests, which would affect addEntry latency. Goals: The main purpose of this feature is to have efficient Garbagecollection story by minimizing the amount of compactions required and the ability to reclaim the deleted ledger's space quicker. Also with this feature we can lay foreground for switching to InterleavedLedgerStorage from SortedLedgerStorage and get predictable tail latency. Proposal: So proposal here is to have multiple active entrylogs. Which will help with compaction performance and

make SortedLedgerStorage redundant. Design Overview: - is to have server configuration specifying number of active entry logs per ledgerdir. - for backward compatibility (for existing behaviour) that config can be set to 0. - round-robin method will be used for choosing the active entry log for the current ledger in EntryLogger.addEntry method - if the total number of active entrylogs is more than or equal to number of active ledgers, then we get almost exclusivity - For implementing Round-Robin approach, we need to maintain state information of mapping of ledgerId to SlotId - there will be numberofledgerdirs*numberofactiveentrylogsperledgerdir slots. a slot is mapped to ledgerdir, but the activeentrylog of that slot will be rotated when it reaches the capacity. - By knowing the SlotId we can get the corresponding entryLogId associated to that slot. - If there is no entry for current ledger in the map, then we pick the next in order slot and add the mapping entry to the map. - Since Bookie won't be informed about the writeclose of the ledger, there is no easy way to know when to remove the mapping entry from the map. Considering it is just <long ledgerid, int slotid> mapentry, we may compromise on evicting policy. We can just use some Cache, which has eviction policy, timebased on last access - If a ledgerdir becomes full, then all the slots having entrylogs in that ledgerdir, should become inactive. The existing mappings, mappings of active ledgers to these slots (active entrylogs), should be updated to available active slots. - when ledgerdir becomes writable again, then the slots which were inactive should be made active and become eligible for round-robin distribution - For this feature I need to make changes to checkpoint logic. Currently with BOOKKEEPER-564 change, we are scheduling checkpoint only when current entrylog file is rotated. So we dont call 'flushCurrentLog' when we checkpoint. But for this feature, since there are going to be multiple active entrylogs, scheduling checkpoint when entrylog file is rotated, is not an option. So I need to call flushCurrentLogs when checkpoint is made for every 'flushinterval' period --- *Enrico Olivelli* 2017-05-19T06:49:35.104+0000 [~jujjuri] [~reddycharan18@gmail.com] This sound very interesting. Now I can see clearly way JV wrote on the mailing list that maybe clients could send some hint to the bookies that a ledger has been gracefully deleted/closed --- *Charan Reddy Guttapalem* 2017-06-02T00:22:59.045+0000 [~sijie@apache.org] created writeup for this work item and discussed about it in last community call (May 18th)

22. **title:** Multiple active entrylogs
    body: JIRA: https://issues.apache.org/jira/browse/BOOKKEEPER-1041 Reporter: Venkateswararao Jujjuri (JV) @jvrao Current bookkeeper is tuned for rotational HDDs. It has one active entrylog, and all the ledger/entries go to the same entrylog until it is rotated out. This is perfect for HDDs as seeks and moving head allover the disk platter is very expensive. But this is very inefficient for SSDs, as each SSD can handle multiple parallel writers, also this method is extremely inefficient for compaction as it causes write amplification and inefficient disk space usage. Our proposal is to have multiple active entrylogs and a configuration param on how many parallel entrylogs the system can have. This way one can have ability to configure to have less (may be one) ledger per entrylog. ### Comments from JIRA --- *Enrico Olivelli* 2017-04-20T07:28:28.619+0000 {quote} But this is very inefficient for HDDs {quote} [~jujjuri] did you mean SSD ? --- *Venkateswararao Jujjuri (JV)* 2017-04-21T16:42:29.203+0000 Yes I mean SSD. Corrected. Thanks. --- *Charan Reddy Guttapalem* 2017-05-18T15:01:22.566+0000 Issue: In Bookie's EntryLogger, we are having only one current active entryLog and all the ledger/entries go to the same entrylog. This is perfect for HDDs as file syncs, seeks and moving head allover the disk platter is very expensive. But having single active Entry Log is inefficient for SSDs, as each SSD can handle multiple parallel writers. Also, having single active EntryLog (irrespective of LedgerStorage type - interleaved/sorted), is inefficient for compaction, since entries of multiple ledgers will end up in an entrylog. Also in SortedLedgerStorage , in the addEntry request we flush EntryMemtable, if it reaches the sizelimit. Because of this we are observing unpredictable tail latency for addEntry request. When EntryMemTable snapshot of size (64 MB) is flushed all at once, this may affect the journal addentry latency. Also, if the rate of new add requests surpasses the rate at which the EntryMemTable's previous snapshot is flushed, then at a point the current EntryMemTable map will reach the limit and since the previous snapshot flush is in progress, EntryMemTable will throttle new addRequests, which would affect addEntry latency. Goals: The main purpose of this feature is to have efficient Garbagecollection story by minimizing the amount of compactions required and the ability to reclaim the deleted ledger's space quicker. Also with this feature we can lay foreground for switching to InterleavedLedgerStorage from SortedLedgerStorage and get predictable tail latency. Proposal: So proposal here is to have multiple active entrylogs. Which will help with compaction performance and make SortedLedgerStorage redundant. Design Overview: - is to have server configuration specifying number of active entry logs per ledgerdir. - for backward compatibility (for existing behaviour) that config can be set to 0. - round-robin method will be used for choosing the active entry log for the current ledger in EntryLogger.addEntry method - if the total number of active entrylogs is more than or equal to number of active ledgers, then we get almost exclusivity - For implementing Round-Robin approach, we need to maintain state information of mapping of ledgerId to SlotId - there will be numberofledgerdirs*numberofactiveentrylogsperledgerdir slots. a slot is mapped to ledgerdir, but the activeentrylog of that slot will be rotated when it reaches the capacity. - By knowing the SlotId we can get the corresponding entryLogId associated to that slot. - If there is no entry for current ledger in the map, then we pick the next in order slot and add the mapping entry to the map. - Since Bookie won't be informed about the writeclose of the ledger, there is no easy way to know when to remove the mapping entry from the map. Considering it is just <long ledgerid, int slotid> mapentry, we may compromise on evicting policy. We can just use some Cache, which has eviction policy, timebased on last access - If a ledgerdir becomes full, then all the slots having entrylogs in that ledgerdir, should become inactive. The existing mappings, mappings of active ledgers to these slots (active entrylogs), should be updated to available active slots. - when ledgerdir becomes writable again, then the slots which were inactive should be made active and become eligible for round-robin distribution - For this feature I need to make changes to checkpoint logic. Currently with BOOKKEEPER-564 change, we are scheduling checkpoint only when current entrylog file is rotated. So we dont call 'flushCurrentLog' when we checkpoint. But for this feature, since there are going to be multiple active entrylogs, scheduling checkpoint when entrylog file is rotated, is not an option. So I need to call flushCurrentLogs when checkpoint is made for every 'flushinterval' period --- *Enrico Olivelli* 2017-05-19T06:49:35.104+0000 [~jujjuri] [~reddycharan18@gmail.com] This sound very interesting. Now I can see clearly way JV wrote on the mailing list that maybe clients could send some hint to the bookies that a ledger has been gracefully deleted/closed --- *Charan Reddy Guttapalem* 2017-06-02T00:22:59.045+0000 [~sijie@apache.org] created writeup for this work item and discussed about it in last community call (May 18th)
23. **title:** Multiple active entrylogs
    body: JIRA: https://issues.apache.org/jira/browse/BOOKKEEPER-1041 Reporter: Venkateswararao Jujjuri (JV) @jvrao Current bookkeeper is tuned for rotational HDDs. It has one active entrylog, and all the ledger/entries go to the same entrylog until it is rotated out. This is perfect for HDDs as seeks and moving head allover the disk platter is very expensive. But this is very inefficient for SSDs, as each SSD can handle multiple parallel writers, also this method is extremely inefficient for compaction as it causes write amplification and inefficient disk space usage. Our proposal is to have multiple active entrylogs and a configuration param on how many parallel entrylogs the system can have. This way one can have ability to configure to have less (may be one) ledger per entrylog. ### Comments from JIRA --- *Enrico Olivelli* 2017-04-20T07:28:28.619+0000 {quote} But this is very inefficient for HDDs {quote} [~jujjuri] did you mean SSD ? --- *Venkateswararao Jujjuri (JV)* 2017-04-21T16:42:29.203+0000 Yes I mean SSD. Corrected. Thanks. --- *Charan Reddy Guttapalem* 2017-05-18T15:01:22.566+0000 Issue: In Bookie's EntryLogger, we are having only one current active entryLog and all the ledger/entries go to the same entrylog. This is perfect for HDDs as file syncs, seeks and moving head allover the disk platter is very expensive. But having single active Entry Log is inefficient for SSDs, as each SSD can handle multiple parallel writers. Also, having single active EntryLog (irrespective of LedgerStorage type - interleaved/sorted), is inefficient for compaction, since entries of multiple ledgers will end up in an entrylog. Also in SortedLedgerStorage , in the addEntry request we flush EntryMemtable, if it reaches the sizelimit. Because of this we are observing unpredictable tail latency for addEntry request. When EntryMemTable snapshot of size (64 MB) is flushed all at once, this may affect the journal addentry latency. Also, if the rate of new add requests surpasses the rate at which the EntryMemTable's previous snapshot

is flushed, then at a point the current EntryMemTable map will reach the limit and since the previous snapshot flush is in progress, EntryMemTable will throttle new addRequests, which would affect addEntry latency. Goals: The main purpose of this feature is to have efficient Garbagecollection story by minimizing the amount of compactions required and the ability to reclaim the deleted ledger's space quicker. Also with this feature we can lay foreground for switching to InterleavedLedgerStorage from SortedLedgerStorage and get predictable tail latency. Proposal: So proposal here is to have multiple active entrylogs. Which will help with compaction performance and make SortedLedgerStorage redundant. Design Overview: - is to have server configuration specifying number of active entry logs per ledgerdir. - for backward compatibility (for existing behaviour) that config can be set to 0. - round-robin method will be used for choosing the active entry log for the current ledger in EntryLogger.addEntry method - if the total number of active entrylogs is more than or equal to number of active ledgers, then we get almost exclusivity - For implementing Round-Robin approach, we need to maintain state information of mapping of ledgerId to SlotId - there will be numberofledgerdirs*numberofactiveentrylogsperledgerdir slots. a slot is mapped to ledgerdir, but the activeentrylog of that slot will be rotated when it reaches the capacity. - By knowing the SlotId we can get the corresponding entryLogId associated to that slot. - If there is no entry for current ledger in the map, then we pick the next in order slot and add the mapping entry to the map. - Since Bookie won't be informed about the writeclose of the ledger, there is no easy way to know when to remove the mapping entry from the map. Considering it is just <long ledgerid, int slotid> mapentry, we may compromise on evicting policy. We can just use some Cache, which has eviction policy, timebased on last access - If a ledgerdir becomes full, then all the slots having entrylogs in that ledgerdir, should become inactive. The existing mappings, mappings of active ledgers to these slots (active entrylogs), should be updated to available active slots. - when ledgerdir becomes writable again, then the slots which were inactive should be made active and become eligible for round-robin distribution - For this feature I need to make changes to checkpoint logic. Currently with BOOKKEEPER-564 change, we are scheduling checkpoint only when current entrylog file is rotated. So we dont call 'flushCurrentLog' when we checkpoint. But for this feature, since there are going to be multiple active entrylogs, scheduling checkpoint when entrylog file is rotated, is not an option. So I need to call flushCurrentLogs when checkpoint is made for every 'flushinterval' period --- *Enrico Olivelli* 2017-05-19T06:49:35.104+0000 [~jujjuri] [~reddycharan18@gmail.com] This sound very interesting. Now I can see clearly way JV wrote on the mailing list that maybe clients could send some hint to the bookies that a ledger has been gracefully deleted/closed --- *Charan Reddy Guttapalem* 2017-06-02T00:22:59.045+0000 [~sijie@apache.org] created writeup for this work item and discussed about it in last community call (May 18th)
label: code-design

24. **title:** Multiple active entrylogs
body: JIRA: https://issues.apache.org/jira/browse/BOOKKEEPER-1041 Reporter: Venkateswararao Jujjuri (JV) @jvrao Current bookkeeper is tuned for rotational HDDs. It has one active entrylog, and all the ledger/entries go to the same entrylog until it is rotated out. This is perfect for HDDs as seeks and moving head allover the disk platter is very expensive. But this is very inefficient for SSDs, as each SSD can handle multiple parallel writers, also this method is extremely inefficient for compaction as it causes write amplification and inefficient disk space usage. Our proposal is to have multiple active entrylogs and a configuration param on how many parallel entrylogs the system can have. This way one can have ability to configure to have less (may be one) ledger per entrylog. ### Comments from JIRA --- *Enrico Olivelli* 2017-04-20T07:28:28.619+0000 {quote} But this is very inefficient for HDDs {quote} [~jujjuri] did you mean SSD ? --- *Venkateswararao Jujjuri (JV)* 2017-04-21T16:42:29.203+0000 Yes I mean SSD. Corrected. Thanks. --- *Charan Reddy Guttapalem* 2017-05-18T15:01:22.566+0000 Issue: In Bookie's EntryLogger, we are having only one current active entryLog and all the ledger/entries go to the same entrylog. This is perfect for HDDs as file syncs, seeks and moving head allover the disk platter is very expensive. But having single active Entry Log is inefficient for SSDs, as each SSD can handle multiple parallel writers. Also, having single active EntryLog (irrespective of LedgerStorage type - interleaved/sorted), is inefficient for compaction, since entries of multiple ledgers will end up in an entrylog. Also in SortedLedgerStorage , in the addEntry request we flush EntryMemtable, if it reaches the sizelimit. Because of this we are observing unpredictable tail latency for addEntry request. When EntryMemTable snapshot of size (64 MB) is flushed all at once, this may affect the journal addentry latency. Also, if the rate of new add requests surpasses the rate at which the EntryMemTable's previous snapshot is flushed, then at a point the current EntryMemTable map will reach the limit and since the previous snapshot flush is in progress, EntryMemTable will throttle new addRequests, which would affect addEntry latency. Goals: The main purpose of this feature is to have efficient Garbagecollection story by minimizing the amount of compactions required and the ability to reclaim the deleted ledger's space quicker. Also with this feature we can lay foreground for switching to InterleavedLedgerStorage from SortedLedgerStorage and get predictable tail latency. Proposal: So proposal here is to have multiple active entrylogs. Which will help with compaction performance and make SortedLedgerStorage redundant. Design Overview: - is to have server configuration specifying number of active entry logs per ledgerdir. - for backward compatibility (for existing behaviour) that config can be set to 0. - round-robin method will be used for choosing the active entry log for the current ledger in EntryLogger.addEntry method - if the total number of active entrylogs is more than or equal to number of active ledgers, then we get almost exclusivity - For implementing Round-Robin approach, we need to maintain state information of mapping of ledgerId to SlotId - there will be numberofledgerdirs*numberofactiveentrylogsperledgerdir slots. a slot is mapped to ledgerdir, but the activeentrylog of that slot will be rotated when it reaches the capacity. - By knowing the SlotId we can get the corresponding entryLogId associated to that slot. - If there is no entry for current ledger in the map, then we pick the next in order slot and add the mapping entry to the map. - Since Bookie won't be informed about the writeclose of the ledger, there is no easy way to know when to remove the mapping entry from the map. Considering it is just <long ledgerid, int slotid> mapentry, we may compromise on evicting policy. We can just use some Cache, which has eviction policy, timebased on last access - If a ledgerdir becomes full, then all the slots having entrylogs in that ledgerdir, should become inactive. The existing mappings, mappings of active ledgers to these slots (active entrylogs), should be updated to available active slots. - when ledgerdir becomes writable again, then the slots which were inactive should be made active and become eligible for round-robin distribution - For this feature I need to make changes to checkpoint logic. Currently with BOOKKEEPER-564 change, we are scheduling checkpoint only when current entrylog file is rotated. So we dont call 'flushCurrentLog' when we checkpoint. But for this feature, since there are going to be multiple active entrylogs, scheduling checkpoint when entrylog file is rotated, is not an option. So I need to call flushCurrentLogs when checkpoint is made for every 'flushinterval' period --- *Enrico Olivelli* 2017-05-19T06:49:35.104+0000 [~jujjuri] [~reddycharan18@gmail.com] This sound very interesting. Now I can see clearly way JV wrote on the mailing list that maybe clients could send some hint to the bookies that a ledger has been gracefully deleted/closed --- *Charan Reddy Guttapalem* 2017-06-02T00:22:59.045+0000 [~sijie@apache.org] created writeup for this work item and discussed about it in last community call (May 18th)
25. **title:** Multiple active entrylogs
body: JIRA: https://issues.apache.org/jira/browse/BOOKKEEPER-1041 Reporter: Venkateswararao Jujjuri (JV) @jvrao Current bookkeeper is tuned for rotational HDDs. It has one active entrylog, and all the ledger/entries go to the same entrylog until it is rotated out. This is perfect for HDDs as seeks and moving head allover the disk platter is very expensive. But this is very inefficient for SSDs, as each SSD can handle multiple parallel writers, also this method is extremely inefficient for compaction as it causes write amplification and inefficient disk space usage. Our proposal is to have multiple active entrylogs and a configuration param on how many parallel entrylogs the system can have. This way one can have ability to configure to have less (may be one) ledger per entrylog. ### Comments from JIRA --- *Enrico Olivelli* 2017-04-20T07:28:28.619+0000 {quote} But this is very inefficient for HDDs {quote} [~jujjuri] did you mean SSD ? --- *Venkateswararao Jujjuri (JV)* 2017-04-21T16:42:29.203+0000 Yes I mean SSD. Corrected. Thanks. --- *Charan Reddy Guttapalem* 2017-05-18T15:01:22.566+0000 Issue: In Bookie's EntryLogger, we are having only one current active entryLog and all the ledger/entries

go to the same entrylog. This is perfect for HDDs as file syncs, seeks and moving head allover the disk platter is very expensive. But having single active Entry Log is inefficient for SSDs, as each SSD can handle multiple parallel writers. Also, having single active EntryLog (irrespective of LedgerStorage type - interleaved/sorted), is inefficient for compaction, since entries of multiple ledgers will end up in an entrylog. Also in SortedLedgerStorage , in the addEntry request we flush EntryMemtable, if it reaches the sizelimit. Because of this we are observing unpredictable tail latency for addEntry request. When EntryMemTable snapshot of size (64 MB) is flushed all at once, this may affect the journal addentry latency. Also, if the rate of new add requests surpasses the rate at which the EntryMemTable's previous snapshot is flushed, then at a point the current EntryMemTable map will reach the limit and since the previous snapshot flush is in progress, EntryMemTable will throttle new addRequests, which would affect addEntry latency. Goals: The main purpose of this feature is to have efficient Garbagecollection story by minimizing the amount of compactions required and the ability to reclaim the deleted ledger's space quicker. Also with this feature we can lay foreground for switching to InterleavedLedgerStorage from SortedLedgerStorage and get predictable tail latency. Proposal: So proposal here is to have multiple active entrylogs. Which will help with compaction performance and make SortedLedgerStorage redundant. Design Overview: - is to have server configuration specifying number of active entry logs per ledgerdir. - for backward compatibility (for existing behaviour) that config can be set to 0. - round-robin method will be used for choosing the active entry log for the current ledger in EntryLogger.addEntry method - if the total number of active entrylogs is more than or equal to number of active ledgers, then we get almost exclusivity - For implementing Round-Robin approach, we need to maintain state information of mapping of ledgerId to SlotId - there will be numberofledgerdirs*numberofactiveentrylogsperledgerdir slots. a slot is mapped to ledgerdir, but the activeentrylog of that slot will be rotated when it reaches the capacity. - By knowing the SlotId we can get the corresponding entryLogId associated to that slot. - If there is no entry for current ledger in the map, then we pick the next in order slot and add the mapping entry to the map. - Since Bookie won't be informed about the writeclose of the ledger, there is no easy way to know when to remove the mapping entry from the map. Considering it is just <long ledgerid, int slotid> mapentry, we may compromise on evicting policy. We can just use some Cache, which has eviction policy, timebased on last access - If a ledgerdir becomes full, then all the slots having entrylogs in that ledgerdir, should become inactive. The existing mappings, mappings of active ledgers to these slots (active entrylogs), should be updated to available active slots. - when ledgerdir becomes writable again, then the slots which were inactive should be made active and become eligible for round-robin distribution - For this feature I need to make changes to checkpoint logic. Currently with BOOKKEEPER-564 change, we are scheduling checkpoint only when current entrylog file is rotated. So we dont call 'flushCurrentLog' when we checkpoint. But for this feature, since there are going to be multiple active entrylogs, scheduling checkpoint when entrylog file is rotated, is not an option. So I need to call flushCurrentLogs when checkpoint is made for every 'flushinterval' period --- *Enrico Olivelli* 2017-05-19T06:49:35.104+0000 [~jujjuri] [~reddycharan18@gmail.com] This sound very interesting. Now I can see clearly way JV wrote on the mailing list that maybe clients could send some hint to the bookies that a ledger has been gracefully deleted/closed --- *Charan Reddy Guttapalem* 2017-06-02T00:22:59.045+0000 [~sijie@apache.org] created writeup for this work item and discussed about it in last community call (May 18th)
label: code-design

26. **title:** Multiple active entrylogs
body: JIRA: https://issues.apache.org/jira/browse/BOOKKEEPER-1041 Reporter: Venkateswararao Jujjuri (JV) @jvrao Current bookkeeper is tuned for rotational HDDs. It has one active entrylog, and all the ledger/entries go to the same entrylog until it is rotated out. This is perfect for HDDs as seeks and moving head allover the disk platter is very expensive. But this is very inefficient for SSDs, as each SSD can handle multiple parallel writers, also this method is extremely inefficient for compaction as it causes write amplification and inefficient disk space usage. Our proposal is to have multiple active entrylogs and a configuration param on how many parallel entrylogs the system can have. This way one can have ability to configure to have less (may be one) ledger per entrylog. ### Comments from JIRA --- *Enrico Olivelli* 2017-04-20T07:28:28.619+0000 {quote} But this is very inefficient for HDDs {quote} [~jujjuri] did you mean SSD ? --- *Venkateswararao Jujjuri (JV)* 2017-04-21T16:42:29.203+0000 Yes I mean SSD. Corrected. Thanks. --- *Charan Reddy Guttapalem* 2017-05-18T15:01:22.566+0000 Issue: In Bookie's EntryLogger, we are having only one current active entryLog and all the ledger/entries go to the same entrylog. This is perfect for HDDs as file syncs, seeks and moving head allover the disk platter is very expensive. But having single active Entry Log is inefficient for SSDs, as each SSD can handle multiple parallel writers. Also, having single active EntryLog (irrespective of LedgerStorage type - interleaved/sorted), is inefficient for compaction, since entries of multiple ledgers will end up in an entrylog. Also in SortedLedgerStorage , in the addEntry request we flush EntryMemtable, if it reaches the sizelimit. Because of this we are observing unpredictable tail latency for addEntry request. When EntryMemTable snapshot of size (64 MB) is flushed all at once, this may affect the journal addentry latency. Also, if the rate of new add requests surpasses the rate at which the EntryMemTable's previous snapshot is flushed, then at a point the current EntryMemTable map will reach the limit and since the previous snapshot flush is in progress, EntryMemTable will throttle new addRequests, which would affect addEntry latency. Goals: The main purpose of this feature is to have efficient Garbagecollection story by minimizing the amount of compactions required and the ability to reclaim the deleted ledger's space quicker. Also with this feature we can lay foreground for switching to InterleavedLedgerStorage from SortedLedgerStorage and get predictable tail latency. Proposal: So proposal here is to have multiple active entrylogs. Which will help with compaction performance and make SortedLedgerStorage redundant. Design Overview: - is to have server configuration specifying number of active entry logs per ledgerdir. - for backward compatibility (for existing behaviour) that config can be set to 0. - round-robin method will be used for choosing the active entry log for the current ledger in EntryLogger.addEntry method - if the total number of active entrylogs is more than or equal to number of active ledgers, then we get almost exclusivity - For implementing Round-Robin approach, we need to maintain state information of mapping of ledgerId to SlotId - there will be numberofledgerdirs*numberofactiveentrylogsperledgerdir slots. a slot is mapped to ledgerdir, but the activeentrylog of that slot will be rotated when it reaches the capacity. - By knowing the SlotId we can get the corresponding entryLogId associated to that slot. - If there is no entry for current ledger in the map, then we pick the next in order slot and add the mapping entry to the map. - Since Bookie won't be informed about the writeclose of the ledger, there is no easy way to know when to remove the mapping entry from the map. Considering it is just <long ledgerid, int slotid> mapentry, we may compromise on evicting policy. We can just use some Cache, which has eviction policy, timebased on last access - If a ledgerdir becomes full, then all the slots having entrylogs in that ledgerdir, should become inactive. The existing mappings, mappings of active ledgers to these slots (active entrylogs), should be updated to available active slots. - when ledgerdir becomes writable again, then the slots which were inactive should be made active and become eligible for round-robin distribution - For this feature I need to make changes to checkpoint logic. Currently with BOOKKEEPER-564 change, we are scheduling checkpoint only when current entrylog file is rotated. So we dont call 'flushCurrentLog' when we checkpoint. But for this feature, since there are going to be multiple active entrylogs, scheduling checkpoint when entrylog file is rotated, is not an option. So I need to call flushCurrentLogs when checkpoint is made for every 'flushinterval' period --- *Enrico Olivelli* 2017-05-19T06:49:35.104+0000 [~jujjuri] [~reddycharan18@gmail.com] This sound very interesting. Now I can see clearly way JV wrote on the mailing list that maybe clients could send some hint to the bookies that a ledger has been gracefully deleted/closed --- *Charan Reddy Guttapalem* 2017-06-02T00:22:59.045+0000 [~sijie@apache.org] created writeup for this work item and discussed about it in last community call (May 18th)
27. **title:** Multiple active entrylogs
body: JIRA: https://issues.apache.org/jira/browse/BOOKKEEPER-1041 Reporter: Venkateswararao Jujjuri (JV) @jvrao Current bookkeeper is tuned for rotational HDDs. It has one active entrylog, and all the ledger/entries go to the same entrylog until it is rotated out. This is perfect for HDDs as seeks and moving head allover the disk platter is very expensive. But this is very inefficient for SSDs, as each SSD can

handle multiple parallel writers, also this method is extremely inefficient for compaction as it causes write amplification and inefficient disk space usage. Our proposal is to have multiple active entrylogs and a configuration param on how many parallel entrylogs the system can have. This way one can have ability to configure to have less (may be one) ledger per entrylog. ### Comments from JIRA --- *Enrico Olivelli* 2017-04-20T07:28:28.619+0000 {quote} But this is very inefficient for HDDs {quote} [~jujjuri] did you mean SSD ? --- *Venkateswararao Jujjuri (JV)* 2017-04-21T16:42:29.203+0000 Yes I mean SSD. Corrected. Thanks. --- *Charan Reddy Guttapalem* 2017-05-18T15:01:22.566+0000 Issue: In Bookie's EntryLogger, we are having only one current active entryLog and all the ledger/entries go to the same entrylog. This is perfect for HDDs as file syncs, seeks and moving head allover the disk platter is very expensive. But having single active Entry Log is inefficient for SSDs, as each SSD can handle multiple parallel writers. Also, having single active EntryLog (irrespective of LedgerStorage type - interleaved/sorted), is inefficient for compaction, since entries of multiple ledgers will end up in an entrylog. Also in SortedLedgerStorage , in the addEntry request we flush EntryMemtable, if it reaches the sizelimit. Because of this we are observing unpredictable tail latency for addEntry request. When EntryMemTable snapshot of size (64 MB) is flushed all at once, this may affect the journal addentry latency. Also, if the rate of new add requests surpasses the rate at which the EntryMemTable's previous snapshot is flushed, then at a point the current EntryMemTable map will reach the limit and since the previous snapshot flush is in progress, EntryMemTable will throttle new addRequests, which would affect addEntry latency. Goals: The main purpose of this feature is to have efficient Garbagecollection story by minimizing the amount of compactions required and the ability to reclaim the deleted ledger's space quicker. Also with this feature we can lay foreground for switching to InterleavedLedgerStorage from SortedLedgerStorage and get predictable tail latency. Proposal: So proposal here is to have multiple active entrylogs. Which will help with compaction performance and make SortedLedgerStorage redundant. Design Overview: - is to have server configuration specifying number of active entry logs per ledgerdir. - for backward compatibility (for existing behaviour) that config can be set to 0. - round-robin method will be used for choosing the active entry log for the current ledger in EntryLogger.addEntry method - if the total number of active entrylogs is more than or equal to number of active ledgers, then we get almost exclusivity - For implementing Round-Robin approach, we need to maintain state information of mapping of ledgerId to SlotId - there will be numberofledgerdirs*numberofactiveentrylogsperledgerdir slots. a slot is mapped to ledgerdir, but the activeentrylog of that slot will be rotated when it reaches the capacity. - By knowing the SlotId we can get the corresponding entryLogId associated to that slot. - If there is no entry for current ledger in the map, then we pick the next in order slot and add the mapping entry to the map. - Since Bookie won't be informed about the writeclose of the ledger, there is no easy way to know when to remove the mapping entry from the map. Considering it is just <long ledgerid, int slotid> mapentry, we may compromise on evicting policy. We can just use some Cache, which has eviction policy, timebased on last access - If a ledgerdir becomes full, then all the slots having entrylogs in that ledgerdir, should become inactive. The existing mappings, mappings of active ledgers to these slots (active entrylogs), should be updated to available active slots. - when ledgerdir becomes writable again, then the slots which were inactive should be made active and become eligible for round-robin distribution - For this feature I need to make changes to checkpoint logic. Currently with BOOKKEEPER-564 change, we are scheduling checkpoint only when current entrylog file is rotated. So we dont call 'flushCurrentLog' when we checkpoint. But for this feature, since there are going to be multiple active entrylogs, scheduling checkpoint when entrylog file is rotated, is not an option. So I need to call flushCurrentLogs when checkpoint is made for every 'flushinterval' period --- *Enrico Olivelli* 2017-05-19T06:49:35.104+0000 [~jujjuri] [~reddycharan18@gmail.com] This sound very interesting. Now I can see clearly way JV wrote on the mailing list that maybe clients could send some hint to the bookies that a ledger has been gracefully deleted/closed --- *Charan Reddy Guttapalem* 2017-06-02T00:22:59.045+0000 [~sijie@apache.org] created writeup for this work item and discussed about it in last community call (May 18th)

28. **title:** Multiple active entrylogs
body: JIRA: https://issues.apache.org/jira/browse/BOOKKEEPER-1041 Reporter: Venkateswararao Jujjuri (JV) @jvrao Current bookkeeper is tuned for rotational HDDs. It has one active entrylog, and all the ledger/entries go to the same entrylog until it is rotated out. This is perfect for HDDs as seeks and moving head allover the disk platter is very expensive. But this is very inefficient for SSDs, as each SSD can handle multiple parallel writers, also this method is extremely inefficient for compaction as it causes write amplification and inefficient disk space usage. Our proposal is to have multiple active entrylogs and a configuration param on how many parallel entrylogs the system can have. This way one can have ability to configure to have less (may be one) ledger per entrylog. ### Comments from JIRA --- *Enrico Olivelli* 2017-04-20T07:28:28.619+0000 {quote} But this is very inefficient for HDDs {quote} [~jujjuri] did you mean SSD ? --- *Venkateswararao Jujjuri (JV)* 2017-04-21T16:42:29.203+0000 Yes I mean SSD. Corrected. Thanks. --- *Charan Reddy Guttapalem* 2017-05-18T15:01:22.566+0000 Issue: In Bookie's EntryLogger, we are having only one current active entryLog and all the ledger/entries go to the same entrylog. This is perfect for HDDs as file syncs, seeks and moving head allover the disk platter is very expensive. But having single active Entry Log is inefficient for SSDs, as each SSD can handle multiple parallel writers. Also, having single active EntryLog (irrespective of LedgerStorage type - interleaved/sorted), is inefficient for compaction, since entries of multiple ledgers will end up in an entrylog. Also in SortedLedgerStorage , in the addEntry request we flush EntryMemtable, if it reaches the sizelimit. Because of this we are observing unpredictable tail latency for addEntry request. When EntryMemTable snapshot of size (64 MB) is flushed all at once, this may affect the journal addentry latency. Also, if the rate of new add requests surpasses the rate at which the EntryMemTable's previous snapshot is flushed, then at a point the current EntryMemTable map will reach the limit and since the previous snapshot flush is in progress, EntryMemTable will throttle new addRequests, which would affect addEntry latency. Goals: The main purpose of this feature is to have efficient Garbagecollection story by minimizing the amount of compactions required and the ability to reclaim the deleted ledger's space quicker. Also with this feature we can lay foreground for switching to InterleavedLedgerStorage from SortedLedgerStorage and get predictable tail latency. Proposal: So proposal here is to have multiple active entrylogs. Which will help with compaction performance and make SortedLedgerStorage redundant. Design Overview: - is to have server configuration specifying number of active entry logs per ledgerdir. - for backward compatibility (for existing behaviour) that config can be set to 0. - round-robin method will be used for choosing the active entry log for the current ledger in EntryLogger.addEntry method - if the total number of active entrylogs is more than or equal to number of active ledgers, then we get almost exclusivity - For implementing Round-Robin approach, we need to maintain state information of mapping of ledgerId to SlotId - there will be numberofledgerdirs*numberofactiveentrylogsperledgerdir slots. a slot is mapped to ledgerdir, but the activeentrylog of that slot will be rotated when it reaches the capacity. - By knowing the SlotId we can get the corresponding entryLogId associated to that slot. - If there is no entry for current ledger in the map, then we pick the next in order slot and add the mapping entry to the map. - Since Bookie won't be informed about the writeclose of the ledger, there is no easy way to know when to remove the mapping entry from the map. Considering it is just <long ledgerid, int slotid> mapentry, we may compromise on evicting policy. We can just use some Cache, which has eviction policy, timebased on last access - If a ledgerdir becomes full, then all the slots having entrylogs in that ledgerdir, should become inactive. The existing mappings, mappings of active ledgers to these slots (active entrylogs), should be updated to available active slots. - when ledgerdir becomes writable again, then the slots which were inactive should be made active and become eligible for round-robin distribution - For this feature I need to make changes to checkpoint logic. Currently with BOOKKEEPER-564 change, we are scheduling checkpoint only when current entrylog file is rotated. So we dont call 'flushCurrentLog' when we checkpoint. But for this feature, since there are going to be multiple active entrylogs, scheduling checkpoint when entrylog file is rotated, is not an option. So I need to call flushCurrentLogs when checkpoint is made for every 'flushinterval' period --- *Enrico Olivelli* 2017-05-19T06:49:35.104+0000 [~jujjuri] [~reddycharan18@gmail.com] This sound very interesting. Now I can see clearly way JV wrote on the mailing list that maybe clients could send some hint to the bookies that a ledger has been gracefully deleted/closed --- *Charan Reddy

Guttapalem* 2017-06-02T00:22:59.045+0000 [~sijie@apache.org] created writeup for this work item and discussed about it in last community call (May 18th)

29. **title:** Multiple active entrylogs
    body: JIRA: https://issues.apache.org/jira/browse/BOOKKEEPER-1041 Reporter: Venkateswararao Jujjuri (JV) @jvrao Current bookkeeper is tuned for rotational HDDs. It has one active entrylog, and all the ledger/entries go to the same entrylog until it is rotated out. This is perfect for HDDs as seeks and moving head allover the disk platter is very expensive. But this is very inefficient for SSDs, as each SSD can handle multiple parallel writers, also this method is extremely inefficient for compaction as it causes write amplification and inefficient disk space usage. Our proposal is to have multiple active entrylogs and a configuration param on how many parallel entrylogs the system can have. This way one can have ability to configure to have less (may be one) ledger per entrylog. ### Comments from JIRA --- *Enrico Olivelli* 2017-04-20T07:28:28.619+0000 {quote} But this is very inefficient for HDDs {quote} [~jujjuri] did you mean SSD ? --- *Venkateswararao Jujjuri (JV)* 2017-04-21T16:42:29.203+0000 Yes I mean SSD. Corrected. Thanks. --- *Charan Reddy Guttapalem* 2017-05-18T15:01:22.566+0000 Issue: In Bookie's EntryLogger, we are having only one current active entryLog and all the ledger/entries go to the same entrylog. This is perfect for HDDs as file syncs, seeks and moving head allover the disk platter is very expensive. But having single active Entry Log is inefficient for SSDs, as each SSD can handle multiple parallel writers. Also, having single active EntryLog (irrespective of LedgerStorage type - interleaved/sorted), is inefficient for compaction, since entries of multiple ledgers will end up in an entrylog. Also in SortedLedgerStorage , in the addEntry request we flush EntryMemtable, if it reaches the sizelimit. Because of this we are observing unpredictable tail latency for addEntry request. When EntryMemTable snapshot of size (64 MB) is flushed all at once, this may affect the journal addentry latency. Also, if the rate of new add requests surpasses the rate at which the EntryMemTable's previous snapshot is flushed, then at a point the current EntryMemTable map will reach the limit and since the previous snapshot flush is in progress, EntryMemTable will throttle new addRequests, which would affect addEntry latency. Goals: The main purpose of this feature is to have efficient Garbagecollection story by minimizing the amount of compactions required and the ability to reclaim the deleted ledger's space quicker. Also with this feature we can lay foreground for switching to InterleavedLedgerStorage from SortedLedgerStorage and get predictable tail latency. Proposal: So proposal here is to have multiple active entrylogs. Which will help with compaction performance and make SortedLedgerStorage redundant. Design Overview: - is to have server configuration specifying number of active entry logs per ledgerdir. - for backward compatibility (for existing behaviour) that config can be set to 0. - round-robin method will be used for choosing the active entry log for the current ledger in EntryLogger.addEntry method - if the total number of active entrylogs is more than or equal to number of active ledgers, then we get almost exclusivity - For implementing Round-Robin approach, we need to maintain state information of mapping of ledgerId to SlotId - there will be numberofledgerdirs*numberofactiveentrylogsperledgerdir slots. a slot is mapped to ledgerdir, but the activeentrylog of that slot will be rotated when it reaches the capacity. - By knowing the SlotId we can get the corresponding entryLogId associated to that slot. - If there is no entry for current ledger in the map, then we pick the next in order slot and add the mapping entry to the map. - Since Bookie won't be informed about the writeclose of the ledger, there is no easy way to know when to remove the mapping entry from the map. Considering it is just <long ledgerid, int slotid> mapentry, we may compromise on evicting policy. We can just use some Cache, which has eviction policy, timebased on last access - If a ledgerdir becomes full, then all the slots having entrylogs in that ledgerdir, should become inactive. The existing mappings, mappings of active ledgers to these slots (active entrylogs), should be updated to available active slots. - when ledgerdir becomes writable again, then the slots which were inactive should be made active and become eligible for round-robin distribution - For this feature I need to make changes to checkpoint logic. Currently with BOOKKEEPER-564 change, we are scheduling checkpoint only when current entrylog file is rotated. So we dont call 'flushCurrentLog' when we checkpoint. But for this feature, since there are going to be multiple active entrylogs, scheduling checkpoint when entrylog file is rotated, is not an option. So I need to call flushCurrentLogs when checkpoint is made for every 'flushinterval' period --- *Enrico Olivelli* 2017-05-19T06:49:35.104+0000 [~jujjuri] [~reddycharan18@gmail.com] This sound very interesting. Now I can see clearly way JV wrote on the mailing list that maybe clients could send some hint to the bookies that a ledger has been gracefully deleted/closed --- *Charan Reddy Guttapalem* 2017-06-02T00:22:59.045+0000 [~sijie@apache.org] created writeup for this work item and discussed about it in last community call (May 18th)
    label: code-design
30. **title:** Multiple active entrylogs
    body: JIRA: https://issues.apache.org/jira/browse/BOOKKEEPER-1041 Reporter: Venkateswararao Jujjuri (JV) @jvrao Current bookkeeper is tuned for rotational HDDs. It has one active entrylog, and all the ledger/entries go to the same entrylog until it is rotated out. This is perfect for HDDs as seeks and moving head allover the disk platter is very expensive. But this is very inefficient for SSDs, as each SSD can handle multiple parallel writers, also this method is extremely inefficient for compaction as it causes write amplification and inefficient disk space usage. Our proposal is to have multiple active entrylogs and a configuration param on how many parallel entrylogs the system can have. This way one can have ability to configure to have less (may be one) ledger per entrylog. ### Comments from JIRA --- *Enrico Olivelli* 2017-04-20T07:28:28.619+0000 {quote} But this is very inefficient for HDDs {quote} [~jujjuri] did you mean SSD ? --- *Venkateswararao Jujjuri (JV)* 2017-04-21T16:42:29.203+0000 Yes I mean SSD. Corrected. Thanks. --- *Charan Reddy Guttapalem* 2017-05-18T15:01:22.566+0000 Issue: In Bookie's EntryLogger, we are having only one current active entryLog and all the ledger/entries go to the same entrylog. This is perfect for HDDs as file syncs, seeks and moving head allover the disk platter is very expensive. But having single active Entry Log is inefficient for SSDs, as each SSD can handle multiple parallel writers. Also, having single active EntryLog (irrespective of LedgerStorage type - interleaved/sorted), is inefficient for compaction, since entries of multiple ledgers will end up in an entrylog. Also in SortedLedgerStorage , in the addEntry request we flush EntryMemtable, if it reaches the sizelimit. Because of this we are observing unpredictable tail latency for addEntry request. When EntryMemTable snapshot of size (64 MB) is flushed all at once, this may affect the journal addentry latency. Also, if the rate of new add requests surpasses the rate at which the EntryMemTable's previous snapshot is flushed, then at a point the current EntryMemTable map will reach the limit and since the previous snapshot flush is in progress, EntryMemTable will throttle new addRequests, which would affect addEntry latency. Goals: The main purpose of this feature is to have efficient Garbagecollection story by minimizing the amount of compactions required and the ability to reclaim the deleted ledger's space quicker. Also with this feature we can lay foreground for switching to InterleavedLedgerStorage from SortedLedgerStorage and get predictable tail latency. Proposal: So proposal here is to have multiple active entrylogs. Which will help with compaction performance and make SortedLedgerStorage redundant. Design Overview: - is to have server configuration specifying number of active entry logs per ledgerdir. - for backward compatibility (for existing behaviour) that config can be set to 0. - round-robin method will be used for choosing the active entry log for the current ledger in EntryLogger.addEntry method - if the total number of active entrylogs is more than or equal to number of active ledgers, then we get almost exclusivity - For implementing Round-Robin approach, we need to maintain state information of mapping of ledgerId to SlotId - there will be numberofledgerdirs*numberofactiveentrylogsperledgerdir slots. a slot is mapped to ledgerdir, but the activeentrylog of that slot will be rotated when it reaches the capacity. - By knowing the SlotId we can get the corresponding entryLogId associated to that slot. - If there is no entry for current ledger in the map, then we pick the next in order slot and add the mapping entry to the map. - Since Bookie won't be informed about the writeclose of the ledger, there is no easy way to know when to remove the mapping entry from the map. Considering it is just <long ledgerid, int slotid> mapentry, we may compromise on evicting policy. We can just use some Cache, which has eviction policy, timebased on last access - If a ledgerdir becomes full, then all the slots having entrylogs in that ledgerdir, should become inactive. The existing mappings, mappings of active ledgers to these slots (active entrylogs), should be updated to available active slots. - when ledgerdir becomes writable again, then the slots which were inactive should be made active and become

eligible for round-robin distribution - For this feature I need to make changes to checkpoint logic. Currently with BOOKKEEPER-564 change, we are scheduling checkpoint only when current entrylog file is rotated. So we dont call 'flushCurrentLog' when we checkpoint. But for this feature, since there are going to be multiple active entrylogs, scheduling checkpoint when entrylog file is rotated, is not an option. So I need to call flushCurrentLogs when checkpoint is made for every 'flushinterval' period --- *Enrico Olivelli* 2017-05-19T06:49:35.104+0000 [~jujjuri] [~reddycharan18@gmail.com] This sound very interesting. Now I can see clearly way JV wrote on the mailing list that maybe clients could send some hint to the bookies that a ledger has been gracefully deleted/closed --- *Charan Reddy Guttapalem* 2017-06-02T00:22:59.045+0000 [~sijie@apache.org] created writeup for this work item and discussed about it in last community call (May 18th)

31. **title:** Multiple active entrylogs
    body: JIRA: https://issues.apache.org/jira/browse/BOOKKEEPER-1041 Reporter: Venkateswararao Jujjuri (JV) @jvrao Current bookkeeper is tuned for rotational HDDs. It has one active entrylog, and all the ledger/entries go to the same entrylog until it is rotated out. This is perfect for HDDs as seeks and moving head allover the disk platter is very expensive. But this is very inefficient for SSDs, as each SSD can handle multiple parallel writers, also this method is extremely inefficient for compaction as it causes write amplification and inefficient disk space usage. Our proposal is to have multiple active entrylogs and a configuration param on how many parallel entrylogs the system can have. This way one can have ability to configure to have less (may be one) ledger per entrylog. ### Comments from JIRA --- *Enrico Olivelli* 2017-04-20T07:28:28.619+0000 {quote} But this is very inefficient for HDDs {quote} [~jujjuri] did you mean SSD ? --- *Venkateswararao Jujjuri (JV)* 2017-04-21T16:42:29.203+0000 Yes I mean SSD. Corrected. Thanks. --- *Charan Reddy Guttapalem* 2017-05-18T15:01:22.566+0000 Issue: In Bookie's EntryLogger, we are having only one current active entryLog and all the ledger/entries go to the same entrylog. This is perfect for HDDs as file syncs, seeks and moving head allover the disk platter is very expensive. But having single active Entry Log is inefficient for SSDs, as each SSD can handle multiple parallel writers. Also, having single active EntryLog (irrespective of LedgerStorage type - interleaved/sorted), is inefficient for compaction, since entries of multiple ledgers will end up in an entrylog. Also in SortedLedgerStorage , in the addEntry request we flush EntryMemtable, if it reaches the sizelimit. Because of this we are observing unpredictable tail latency for addEntry request. When EntryMemTable snapshot of size (64 MB) is flushed all at once, this may affect the journal addentry latency. Also, if the rate of new add requests surpasses the rate at which the EntryMemTable's previous snapshot is flushed, then at a point the current EntryMemTable map will reach the limit and since the previous snapshot flush is in progress, EntryMemTable will throttle new addRequests, which would affect addEntry latency. Goals: The main purpose of this feature is to have efficient Garbagecollection story by minimizing the amount of compactions required and the ability to reclaim the deleted ledger's space quicker. Also with this feature we can lay foreground for switching to InterleavedLedgerStorage from SortedLedgerStorage and get predictable tail latency. Proposal: So proposal here is to have multiple active entrylogs. Which will help with compaction performance and make SortedLedgerStorage redundant. Design Overview: - is to have server configuration specifying number of active entry logs per ledgerdir. - for backward compatibility (for existing behaviour) that config can be set to 0. - round-robin method will be used for choosing the active entry log for the current ledger in EntryLogger.addEntry method - if the total number of active entrylogs is more than or equal to number of active ledgers, then we get almost exclusivity - For implementing Round-Robin approach, we need to maintain state information of mapping of ledgerId to SlotId - there will be numberofledgerdirs*numberofactiveentrylogsperledgerdir slots. a slot is mapped to ledgerdir, but the activeentrylog of that slot will be rotated when it reaches the capacity. - By knowing the SlotId we can get the corresponding entryLogId associated to that slot. - If there is no entry for current ledger in the map, then we pick the next in order slot and add the mapping entry to the map. - Since Bookie won't be informed about the writeclose of the ledger, there is no easy way to know when to remove the mapping entry from the map. Considering it is just <long ledgerid, int slotid> mapentry, we may compromise on evicting policy. We can just use some Cache, which has eviction policy, timebased on last access - If a ledgerdir becomes full, then all the slots having entrylogs in that ledgerdir, should become inactive. The existing mappings, mappings of active ledgers to these slots (active entrylogs), should be updated to available active slots. - when ledgerdir becomes writable again, then the slots which were inactive should be made active and become eligible for round-robin distribution - For this feature I need to make changes to checkpoint logic. Currently with BOOKKEEPER-564 change, we are scheduling checkpoint only when current entrylog file is rotated. So we dont call 'flushCurrentLog' when we checkpoint. But for this feature, since there are going to be multiple active entrylogs, scheduling checkpoint when entrylog file is rotated, is not an option. So I need to call flushCurrentLogs when checkpoint is made for every 'flushinterval' period --- *Enrico Olivelli* 2017-05-19T06:49:35.104+0000 [~jujjuri] [~reddycharan18@gmail.com] This sound very interesting. Now I can see clearly way JV wrote on the mailing list that maybe clients could send some hint to the bookies that a ledger has been gracefully deleted/closed --- *Charan Reddy Guttapalem* 2017-06-02T00:22:59.045+0000 [~sijie@apache.org] created writeup for this work item and discussed about it in last community call (May 18th)
    label: code-design
32. **title:** Multiple active entrylogs
    body: JIRA: https://issues.apache.org/jira/browse/BOOKKEEPER-1041 Reporter: Venkateswararao Jujjuri (JV) @jvrao Current bookkeeper is tuned for rotational HDDs. It has one active entrylog, and all the ledger/entries go to the same entrylog until it is rotated out. This is perfect for HDDs as seeks and moving head allover the disk platter is very expensive. But this is very inefficient for SSDs, as each SSD can handle multiple parallel writers, also this method is extremely inefficient for compaction as it causes write amplification and inefficient disk space usage. Our proposal is to have multiple active entrylogs and a configuration param on how many parallel entrylogs the system can have. This way one can have ability to configure to have less (may be one) ledger per entrylog. ### Comments from JIRA --- *Enrico Olivelli* 2017-04-20T07:28:28.619+0000 {quote} But this is very inefficient for HDDs {quote} [~jujjuri] did you mean SSD ? --- *Venkateswararao Jujjuri (JV)* 2017-04-21T16:42:29.203+0000 Yes I mean SSD. Corrected. Thanks. --- *Charan Reddy Guttapalem* 2017-05-18T15:01:22.566+0000 Issue: In Bookie's EntryLogger, we are having only one current active entryLog and all the ledger/entries go to the same entrylog. This is perfect for HDDs as file syncs, seeks and moving head allover the disk platter is very expensive. But having single active Entry Log is inefficient for SSDs, as each SSD can handle multiple parallel writers. Also, having single active EntryLog (irrespective of LedgerStorage type - interleaved/sorted), is inefficient for compaction, since entries of multiple ledgers will end up in an entrylog. Also in SortedLedgerStorage , in the addEntry request we flush EntryMemtable, if it reaches the sizelimit. Because of this we are observing unpredictable tail latency for addEntry request. When EntryMemTable snapshot of size (64 MB) is flushed all at once, this may affect the journal addentry latency. Also, if the rate of new add requests surpasses the rate at which the EntryMemTable's previous snapshot is flushed, then at a point the current EntryMemTable map will reach the limit and since the previous snapshot flush is in progress, EntryMemTable will throttle new addRequests, which would affect addEntry latency. Goals: The main purpose of this feature is to have efficient Garbagecollection story by minimizing the amount of compactions required and the ability to reclaim the deleted ledger's space quicker. Also with this feature we can lay foreground for switching to InterleavedLedgerStorage from SortedLedgerStorage and get predictable tail latency. Proposal: So proposal here is to have multiple active entrylogs. Which will help with compaction performance and make SortedLedgerStorage redundant. Design Overview: - is to have server configuration specifying number of active entry logs per ledgerdir. - for backward compatibility (for existing behaviour) that config can be set to 0. - round-robin method will be used for choosing the active entry log for the current ledger in EntryLogger.addEntry method - if the total number of active entrylogs is more than or equal to number of active ledgers, then we get almost exclusivity - For implementing Round-Robin approach, we need to maintain state information of mapping of ledgerId to SlotId - there will be numberofledgerdirs*numberofactiveentrylogsperledgerdir slots. a slot is mapped to ledgerdir, but the activeentrylog of that slot will be rotated when it reaches the capacity. - By knowing the SlotId we can get the corresponding

entryLogId associated to that slot. - If there is no entry for current ledger in the map, then we pick the next in order slot and add the mapping entry to the map. - Since Bookie won't be informed about the writeclose of the ledger, there is no easy way to know when to remove the mapping entry from the map. Considering it is just <long ledgerid, int slotid> mapentry, we may compromise on evicting policy. We can just use some Cache, which has eviction policy, timebased on last access - If a ledgerdir becomes full, then all the slots having entrylogs in that ledgerdir, should become inactive. The existing mappings, mappings of active ledgers to these slots (active entrylogs), should be updated to available active slots. - when ledgerdir becomes writable again, then the slots which were inactive should be made active and become eligible for round-robin distribution - For this feature I need to make changes to checkpoint logic. Currently with BOOKKEEPER-564 change, we are scheduling checkpoint only when current entrylog file is rotated. So we dont call 'flushCurrentLog' when we checkpoint. But for this feature, since there are going to be multiple active entrylogs, scheduling checkpoint when entrylog file is rotated, is not an option. So I need to call flushCurrentLogs when checkpoint is made for every 'flushinterval' period --- *Enrico Olivelli* 2017-05-19T06:49:35.104+0000 [~jujjuri] [~reddycharan18@gmail.com] This sound very interesting. Now I can see clearly way JV wrote on the mailing list that maybe clients could send some hint to the bookies that a ledger has been gracefully deleted/closed --- *Charan Reddy Guttapalem* 2017-06-02T00:22:59.045+0000 [~sijie@apache.org] created writeup for this work item and discussed about it in last community call (May 18th)

github_issues_comments:

1. **body:** For this work item, there are two possible approaches 1) is to have configurable number of active entrylogs per ledgerdir 2) active entrylog per ledger For the first approach I mentioned the design overview earlier. But we decided to go with the second approach for the following reasons There are couple of major benefits by going with entrylogperledger. The implementation would become simpler, because it is a complex logic to handle and maintain slotmap in configured number of Entrylogs per LedgerDir design/implementation and it needs to be thoroughly analyzed/tested from multi-thread perspective. But most importantly the garbagecollection component becomes much simpler because there is no need of compaction in entrylog per ledger approach. So there is huge gain in indirect performance because of absence of IO activity for compaction and huge improvement in space reclamation during compaction.
   label: code-design
2. **Design Overview** **Entrylog per Ledger** As name suggests, in this approach in a bookie we are going to have active entrylog dedicated to active ledger. But it is not a strict enforcement of one-to-one mapping of ledger to entrylog in a bookie. Strict one-to-one relationship between ledger and entrylog in a bookie is not possible for multiple reasons (ledgerdir might become full, entrylog might reach its capacity, segmentation/replication can happen, possibility of bookie crash,..). Besides, once entrylog is rotated it cannt be reopened. Since while rotating entrylog file, EntryLogger appends the ledger map at the end of the entry log and updates the entrylog file with the offset and size of the map. It is like sealing the entrylog file. And EntryLogger maintains a pointer called 'leastUnflushedLogId', which specifies the least entrylogid which is not yet rotated and closed and GC considers all the entrylogs with logid lesser than 'leastUnflushedLogId' (entrylogids are sequential numbers) are eligible for compaction/garbagecollection. In summary once the entrylog is rotated and closed we need to maintain immutable semantics on entrylog file. So instead we can provide relaxed constraint where an entrylog is dedicated/committed to a ledger but not otherway around. So in most cases there would be just one entrylog for a ledger in a bookie, but in situations like when entrylog reaches capacity it is rotated and new one is created for that ledger, when ledgerdir is full all the entrylogs in that ledgerdir are rotated and new ones are created for those ledgers, because of segmentation and replication various segments of ledger might end up in different entrylogs and because of a bookie crash while replaying the journal new entrylog will be created for the leftout entries in the journal., entries of a ledger might end up in different entrylogs in a bookie. To summarize briefly about this approach. - is to have server configuration specifying entrylogperledger is enabled - for the previous behavior (one active entrylog) that config can be set to false - when entrylogger receives addEntry call, it needs to know the entry log for the current ledger - so EntryLogger needs to maintain state information of mapping of ledgerId to entrylogid. If the in-memory map doesn't contain entry for the ledger, then EntryLogger will create a new Entrylog and add the mapping of ledgerId to EntryLog. - for creation of new entrylog, EntryLogger will pick writable ledgerdir with least number of active entrylogs - if entrylog reaches the capacity, then it will be rotated and new entrylog will be assigned to that ledger and mapping will be updated - If a ledgerdir becomes full, then all the entrylogs in that ledgerdir, should be rotated. New EntryLogs should be created in the available writable ledgerdirs for those ledgers and the mapping should be updated. - when ledgerdir becomes writable again that ledgerdir should become eligible for creation of new entrylogs - Currently Bookie is not informed about the writeclose of the ledger, so there needs to be a way to know when to remove the mapping entry from the map and rotate the entrylog. One simple way to handle it is to use cache (Guava Cache library) with timebased eviction policy (on last access) and as part of removal listener we can rotate the corresponding entrylog. - Time based eviction policy is simple to provide, but untill entrylog file is rotated and flushed, filehandles of entrylogs are kept open and it wont become eligible for compaction/garbage collection. So explicit writeclose call from client to bookies ensemble of that ledger is needed for better handling of entrylogs. - Both the time based eviction and removal policy and explicit writeclose call are required because not in all cases explicit write close calls to bookies are guaranteed, like during ensemble change of ledger, client crash and unreliable write close protocol. Advisory Write Close is explained below in detail. - For this feature I need to make changes to checkpoint logic. Currently with BOOKKEEPER-564 change, we are scheduling checkpoint only when current entrylog file is rotated. So we dont call 'flushCurrentLog' when we checkpoint. But for this feature, since there are going to be multiple active entrylogs, scheduling checkpoint when entrylog file is rotated, is not an option. So I need to call flushCurrentLogs when checkpoint is made for every 'flushinterval' period - With entrylogperledger feature we are not changing format of the entrylog contents in anyway, so it should be possible to switch entrylogperledger configuration back and forth. **Advisory Write Close implementation details** - Advisory write close message should be sent to all the bookies of the current ensemble when ledger is write closed or recover opened. - Client operation will not wait for the callback of its call. - the callback of the advisory close operation is going to be just logger. It just logs message if it is success or log error in case of any error. - bookie should communicate the ledger close message to entrylogger and it should store that message in memory datastructure. - when the next checkpoint happens after flushing all the entries of the ledger to the corresponding entrylog, then it should use the close signal to rotate the corresponding entrylog.
3. > The implementation would become simpler, because it is a complex logic to handle and maintain slotmap in configured number of Entrylogs per LedgerDir design/implementation and it needs to be thoroughly analyzed/tested from multi-thread perspective. But most importantly the garbagecollection component becomes much simpler because there is no need of compaction in entrylog per ledger approach. So there is huge gain in indirect performance because of absence of IO activity for compaction and huge improvement in space reclamation during compaction. This needs to be clarified. It would have the benefits if you have small number of ledgers, I don't think it is good for large number of ledgers. It is good if you think from Salesforce's use case; however it is not, if you think from other use cases. This should be called out before you are claiming all the benefits here.
4. **body:** > Currently with BOOKKEEPER-564 change, we are scheduling checkpoint only when current entrylog file is rotated. So we dont call 'flushCurrentLog' when we checkpoint. But for this feature, since there are going to be multiple active entrylogs, scheduling checkpoint when entrylog file is rotated, is not an option. So I need to call flushCurrentLogs when checkpoint is made for every 'flushinterval' period just to point out, it would be good to show there is no performance penalty, if entrylogperledger is disable. otherwise it would potentially impact all existing use cases where entrylogperledger is ideal in those use cases.
   label: code-design

5. **body:** It would be good if this could be implemented as a new LedgerStorage implementation, rather than modifying what we have already with a bunch of boolean flags scattered around the code. I see the patch makes modifications to EntryMem table for example, which is entirely unnecessary if you only write a single ledger to an entrylog.
   label: code-design
6. **body:** Hey Ivan.. Sijie, JV and our internal team had long conversation about the same and I responded to community aswell in one of the mail Initially when we designed to implement this feature by providing config for number of entrylogs per ledgerdir and because of that the logic was convoluted with slot mapping management and Sijie felt that it is complicated and questioned about it. But now with entrylog per ledger, the management logic in entrylogger is simpler and straightforward. The same amount of complication will be applicable even in the case of composition at LedgerStorage level. When we get addEntry at ComposedLedgerStorage there should be logic to which LedgerStorage it should goto. So with this fact I can say that composing at LedgerStorage level it will be atleast as complicated as composing at EntryLogger level. with the ComposedLedgerStorage approach (Composite Pattern), ComposedLedgerStorage managing N InterleavedLedgerStorage/SortedLedgerStorages, there is going to be considerable change in the way the resources and state information are handled. Each Interleaved/SortedLedgerStorage will have its own EntryLogger (to serve our MultipleEntryLogs feature), but rest all needs to be shared, mainly 'LedgerCache', 'gcThread' and 'activeLedgers'. So there is going to be major churn in the code for moving the operations dealing with shared resources and state to ComposedLedgerStorage and leave the rest in InterleavedLedgerStorage. Amount of changes required in testcode to deal with these changes is even more. We have to do this while providing backward compatibility (Single EntryLog). Now this should make one question where should the composition happen. As far as I can say, it is supposed to happen at the lowest possible level rather than at higher level, which needs extra band-aid efforts to deal separately for the common resources/state and multiplexable resources. with composition at EntryLogger level, currently in my implementation getEntry/readEntry path, index manager and GC components are unchanged. but with composition at LedgerStorage even for readEntry/getEntry the multiplexing/redirection needs to happen. This is dealbreaker for me. Things get very messy with SortedLedgerSotrage entries in Memtable when Ledgerdir of the entryLog becomes full (we have to rotate current entrylog and create new one in writableledgerdir) while simulataneous writes and read are happening. Also this needs instance of LedgerStorage class for each entryLog in readpath and corresponding EntryMemTable if it is SortedLedgerStorage, though Memtable is not needed in read case. Most importantly we need to provide this feature with backward compatibility (we should be able to read previous data) and also provide existing behaviour (single entrylog) with config option. and mainly LedgerStorage is not the only consumer of EntryLogger, but also GarbageCollectorThread. It calls quite a few methods of EntryLogger - (entryLogger.addEntry, entryLogger.flush, entryLogger.removeEntryLog, entryLogger.scanEntryLog, entryLogger.getLeastUnflushedLogId, entryLogger.logExists, entryLogger.getEntryLogMetadata). It is going to be an issue with ComposedLedgerStorage, because with that EntryLogger is going to be responsible for just one EntryLog. For GarbageCollectorThread to work correctly with multiple EntryLogs, composition is required here as well. Which is double whammy when it comes down to implementation. Anyhow the only posiible shortcoming i see with multiplexing at Entrylogger level is that we are having single MemTable in the case of SortedLedgerStorage, but we mitigated that issue by doing parallel flushes. In terms of complexity and code churn if we go with multiplexing at ledgerstorage class level it will be several times higher than the current implementation with not much benefits.
   label: code-design
7. > It would be good if this could be implemented as a new LedgerStorage implementation, rather than modifying what we have already with a bunch of boolean flags scattered around the code. I see the patch makes modifications to EntryMem table for example, which is entirely unnecessary if you only write a single ledger to an entrylog. I think this has been discussed a while back. I would suggest no holding this for requesting a new ledger storage implementation. In order to make the community move forward with this change, I would suggest taking what Charan has at this moment, and review it based on that. We can consider any better refactoring after that.
8. @reddycharan I'm not suggesting to have a ledgerstorage composed of multiple entrylogs. But the it makes no sense to make this modification on InterleavedLedgerStorage. By definition, this change creates a non-interleaved ledger storage. And SortedLedgerStorage and this change should have absolutely nothing to do with each other. Sorting is sorting entries before they flush, so that entries from the same ledger are close to each other. With logger per ledger, this is pointless. The interleaved in ledgerstorage is the entrylogger. What I would suggest is to create a abstract BaseLedgerStorage which holds common code, and derive a LoggerPerLedgerLedgerStorage and InterleavedLedgerStorage from that. The only difference should be the EntryLogger implementation. If garbagecollection needs to change based on the entrylogger implementation, then each should have it's own garbage collection implemention. > I would suggest no holding this for requesting a new ledger storage implementation. In order to make the community move forward with this change, I would suggest taking what Charan has at this moment, and review it based on that. We can consider any better refactoring after that. What is the rush? I've seen this suggestion to just wave through a patch for expedience a couple of times now, and I think it's a terrible practice. It just means we keep accumulating tech debt. I remember wanting to do a refactor once the twitter changes got in. That never happened for a number of reason. Then we discussed a refactor of checkpointing once yahoo changes are in. This patch in itself is 2900 LOC. Unless some takes two whole days to review that, then its not going to get a thorough review before submission. This problem compounds each time the patch is updated. That alone is reason to slow down.
9. **body:** > What is the rush? I am not suggesting a rush. What I am suggesting here is to respect to what have been discussed and agreed on. What I am saying here is that this change has been raised up and discussed a couple of time since a few months ago, and we have sort of settled down about the approach and the implementation we need to take. It doesn't make any sense for re-doing all this discussion and wasting the time we had in the discussion. That's what I am suggesting for not changing the direction on how things have been implemented here, otherwise that means JV, Charan and me have to go through another round of discussion and Charan might have to redo all his changes, which doesn't seems to be worth doing that. Also if you take a closer look at Charan's change, it isn't really too much different from what you suggested in your recent comment. It is just a matter where does the abstraction happen at. His change is at the entry log manager level, which he attempts to provide a entry log manager to abstract how different implementations manage entry log files. what you suggest might be similar but slight different abstraction interface, which from my personal view, there is no real fundamental differences. That's why I suggest that if he can improve his current `EntryLogManager` interface and hide the details on how different implementations managing entry log files. That will be a good implementation to ship this feature. > I've seen this suggestion to just wave through a patch for expedience a couple of times now, and I think it's a terrible practice. I remember wanting to do a refactor once the twitter changes got in. That never happened for a number of reason. Then we discussed a refactor of checkpointing once yahoo changes are in. I think people would have different views on things, such as how to move forward the project, how to implement a feature, how is the code organized. We are in a public project, collaborating in the same project, different organizations might also have different priorities on things. For example, for us, "per entry log" might not be priority, while it might be a priority for salesforce. That is something that we should have to respect to each other. The examples you described here about twitter and yahoo cases also came from the priorities that twitter and yahoo used to have, whether those refactor are needed are also have to based on priorities that each organization has. I am not sure that would be the thing called "terrible practice". from my personal experiences, "shipit" is never a terrible thing to do, and that's how different organizations can really move the project into production and shipit on production and move it forward, rather than being stuck at waiting perfect refactor (which I don't think there is going to be a perfect code refactor). so in my view, "shipit" is much important for a lot of organizations than other things. The tech debts you are mentioning about really only come to true when there are priorities to move them forward. > This patch in itself is 2900 LOC. Unless some takes two whole days to review that, then its not going to get a thorough review before submission. This problem compounds each time the patch is updated. That alone is reason to slow down. That is a real issue for any changes and for any person

reviewing the change. I don't think things are going to change if changing a different approach to implement this. You will probably feel it easy to review, because the implementation is aligned to what you have been thinking in your mind. But the situation might still remain true to other reviewers reviewing the change. I have found it useful if I think from the author's perspective and understand what he was thinking, that would be easier for me to understand his change and know how different people think of the same problem and adjust to coding style according, that would make reviews easier and people can come to agreement much faster.
label: code-design

10. > What I am suggesting here is to respect to what have been discussed and agreed on. Where has this been discussed? I can't find anything on the dev@ list. > His change is at the entry log manager level, which he attempts to provide a entry log manager to abstract how different implementations manage entry log files. what you suggest might be similar but slight different abstraction interface, which from my personal view, there is no real fundamental differences. There's a big difference. Making the change in the entrylogger itself, means that the implementation gets pulled into SortedLedgerStorage, and that means he has to deal with the EntryMemTable, which should have exactly nothing to do with the implementation. > so in my view, "shipit" is much important for a lot of organizations than other things. Shipping is a function of many things, not just getting code into branch. > That is a real issue for any changes and for any person reviewing the change. I don't think things are going to change if changing a different approach to implement this. You will probably feel it easy to review, because the implementation is aligned to what you have been thinking in your mind. 3000 LOC is never easy to review. 300-400 LOC is the max that can be reviewed competently in an hour, any more, people start skimming. I wrote an email about this in october[1], so I won't go through it again. Any way this is implemented, it needs to be broken into smaller patches (ideally 2-300LOC), so that each coherent piece can be thoroughly reviewed, and the reviewer can be sure that what is submitted is getting sufficient test coverage. If the patch cannot be broken, this in itself raises questions about the architecture of it. [1] https://lists.apache.org/thread.html/48db536c7208dcad87451d84e61959b0266cbe3860fcb0dafc451995@%3Cdev.bookkeeper.apache.org%3E
11. **body:** > Where has this been discussed? I can't find anything on the dev@ list. The discussion has been started since May 2017 and it has been mentioned/discussed at different forms in different email threads, different community meeting or direct meetings. http://mail-archives.apache.org/mod_mbox/bookkeeper-dev/201707.mbox/%3CCAAFz1KPYb4f%2BrUfP2Bhvy%2BigksBLsuXE-QMw6U%2B%3DBDiRXQ19LA%40mail.gmail.com%3E http://mail-archives.apache.org/mod_mbox/bookkeeper-dev/201705.mbox/%3CCAAFz1KNr4uB8RMiXK97gkiqfXCVEoygfjeR2rJKcddvF9ZM1yg%40mail.gmail.com%3E > There's a big difference. Making the change in the entrylogger itself, means that the implementation gets pulled into SortedLedgerStorage, and that means he has to deal with the EntryMemTable, which should have exactly nothing to do with the implementation. In order to fully leverage "per ledger" characteristics, parallel flushing is a necessary change. But it doesn't mean the change in EntryMemTable is only dedicated to per ledger entry log manager. It can be useful for single entry log manager. I requested Charan to not enable parallel flushing feature for single entry log manager until we have confident on that change. And that's why there is boolean flags around that piece. This change here is actually taking `SortedLedgerStorage` in a right direction: 1) MemTable does cache and sorting, you can have different flushing policy (sequential vs parallel) 2) EntryLogManager manages entry log files, when to create them, how to rotate them and how to flush/checkpoint them. 3) EntryLogger becomes the entrypoint to append entries, where the complexity of managing files are handed over to entry log manager. This change takes the effort to allow us provide a better modularized implementation of SortedLedgerStorage, which it is the default storage implementation for most of bookkeeper users. > Shipping is a function of many things, not just getting code into branch. I completely agreed with you. and this change is not "getting code into branch" if you have followed the discussion since May last year. > 3000 LOC is never easy to review. 300-400 LOC is the max that can be reviewed competently in an hour, any more, people start skimming. I wrote an email about this in october[1], so I won't go through it again. I agreed with you - smaller patches are much easier to review and get changes in. However this is a very subjective thing that varies on different things and to different people. sometimes reviewers won't see a clear picture if a change is small and they will get confused; reviewers will be overwhelmed if they don't follow closely on the discussion if the change is big. so both cases can exist and this process can not really be quantified. What we have to do is to follow what we called guidelines or practices and encourage people to send changes in small batches. However there can still be exceptions. for example, code changes has been made before (like changes have been made in branches and run on production. for example, the merges come from branches). In this case, it doesn't make any sense for the people cherry-picking the change to redo the changes by breaking down into small patches. I can't speak for all the cases. But for the case here, this feature has been discussed since May last year, and the change has been reviewed/discussed multiple times internally by Salesforce folks and externally by me during this long discussion and review recycle. so I am fine with reviewing the change in its current form.
label: code-design
12. > The discussion has been started since May 2017 and it has been mentioned/discussed at different forms in different email threads, Ah, I didn't go back as far as July. I don't see any concensus on design or a design document though. In fact, the discussions highlighted aren't even about per ledger ledger storage, but a question about synchronization. In fact, I see a lot of the issues I am raising being risen there. > different community meeting There's nothing in the notes about a decision on a design. That's not to say that it wasn't discussed, but there's no record of a design discussion/decision, so it may as well not have happened. The Apache Board highlighted exactly this issue a year ago (feedback on 2017-02-27 report). > direct meetings. As far as the community is concerned, what happens in direct meetings may as well not have happened. > 1) MemTable does cache and sorting, you can have different flushing policy (sequential vs parallel) If this feature is useful in itself, then break it out into a separate change. > 2) EntryLogManager manages entry log files, when to create them, how to rotate them and how to flush/checkpoint them. 3) EntryLogger becomes the entrypoint to append entries, where the complexity of managing files are handed over to entry log manager. EntryLogger creates and owns the entry log manager, so Entrylogger is effectively managing both when viewed from a higher level. > sometimes reviewers won't see a clear picture if a change is small and they will get confused; This is where a design doc is useful. Or push a wip branch, and then break out small pieces. > However there can still be exceptions. for example, code changes has been made before (like changes have been made in branches and run on production. for example, the merges come from branches). In this case, it doesn't make any sense for the people cherry-picking the change to redo the changes by breaking down into small patches. This change modifies a lot of internals which are used by all running bookies. - EntryLogger - EntryMemTable - InterleavedLedgerStorage - SortedLedgerStorage - BufferedChannel A problematic change in any of these means lost data. It's a high risk change, or rather these are all high risk changes, so each one needs to be justified and examined closely. Contrast this to a similar change, bringing rocks db storage into master (https://github.com/apache/bookkeeper/commit/bba1c6f5d0). It's actually a similar size change. But it doesn't touch any existing class, so it's low risk. > I can't speak for all the cases. But for the case here, this feature has been discussed since May last year, and the change has been reviewed/discussed multiple times internally by Salesforce folks and externally by me during this long discussion and review recycle. so I am fine with reviewing the change in its current form. To summarize, I'm -1 on the changes in their current form. This doesn't mean they can't go in obviously, just that you'll need two +1 from elsewhere. My primary concern is the modification of core classes in an uberpatch. A secondary concern is how per ledger ledger storage behaves in the class hierarchy (i.e. it shouldn't touch interleaved).
13. > Ah, I didn't go back as far as July. I don't see any concensus on design or a design document though. In fact, the discussions highlighted aren't even about per ledger ledger storage, but a question about synchronization. In fact, I see a lot of the issues I am raising being risen there. > There's nothing in the notes about a decision on a design. That's not to say that it wasn't discussed, but there's no record of a design discussion/decision, so it may as well not have happened. The Apache Board highlighted exactly this issue a year ago (feedback on 2017-02-27 report). > As far as the community is concerned, what happens in direct meetings may as well not have happened. what I was trying to

explain here is there is already an effort since last May, which I wish you respect to the fact that three people have being working on this topic. I never used "decision" in my comments or said it is a "decision". I am not sure why you are going to pull the whole conversation to "decision" and "ASF policy" which sounds political and doesn't make any sense to me. A final decision for a code change or feature is made through approvals or +1s on the pull requests following the bylaws. This is the easy thing to reason about; all the others are efforts/discussion, which can happen in the community meetings, direct messages/chats, these are records not decisions, which should be kept in ASF infra at their best efforts. for feedback 2017-02-27, I have responded that we have all the meeting notes recorded. The board doesn't have any objections to that. And this is irrelative to the topic/questions here, which I would prefer leaving out of this thread. > To summarize, I'm -1 on the changes in their current form. This doesn't mean they can't go in obviously, just that you'll need two +1 from elsewhere. first of all, I am not the author of this feature. I am not sure why your statements sound like I am pushing this change in. what I have been done in this thread is just to explain the fact to you, my view of current approach. anyway, I have tried to explain what I can explain here, would like to defer any technical questions to Charan, since he is the best person to answer it. regarding the approach and the implementation, I am fine with current approach implemented in the pull request and am okay to continue my review in that form.

14. > I never used "decision" in my comments or said it is a "decision". In your comment yesterday, you said "What I am suggesting here is to respect to what have been discussed and agreed on. ". Agreed on implies a decision.
15. Hey @ivankelly , to answer â€œwhen/where was this discussed partâ€ , I can point you to the meeting notes, mail exchanges, slack messages and git links which I shared with the community throughout the period. This work item was initially created by @jvrao during April 2017 https://issues.apache.org/jira/browse/BOOKKEEPER-1041. And by May 2017 I came up with design overview and implementation choices and presented to the community on May 18th community meeting. You can find the meeting minutes of what I presented on that day - https://cwiki.apache.org/confluence/display/BOOKKEEPER/2017-5-18+Meeting+Notes and I explained the design overview in the same jira issue. And yes, initially the design was to have configured number of entrylogs per ledgerdir. While implementing it, I noticed few issues with how checkpoint is done, regarding synchronization logic and preallocation of entry logs and raised the concerns regarding the same in the community by starting mail threads and having conversations in the community bi-weekly meets. In one of such mail conversation, @sijie asked same set of questions regarding where the abstraction/composition should happen. I explained in detail in the following mail http://mail-archives.apache.org/mod_mbox/bookkeeper-dev/201707.mbox/%3CCAAFz1KNJbPnhLGwp27kKKGgOtLiDPR7FHpSngdcN-C-Njxt7eQ%40mail.gmail.com%3E . Also in this email I provided git link for the initial implementation of code https://github.com/reddycharan/bookkeeper/tree/multipleentrylogs. Following this mail conversation @sijie , @jvrao and I had in length conversation regarding where the composition should happen and convinced @sijie that given the way code is structured it is right thing to do (abstraction/composition) at lowest level - Entrylogger level. But in the follow up, Iâ€™ve been questioned about the intricacies of multithreaded/synchronization aspects of slot map management in configured number of entrylogs per ledgerdir approach and the benefits of having configured number of entrylogs per ledgerdir vs explicit entry log per ledger in the compaction story (since it is binary decision of whether to keep the entry log or not during compaction in the entrylog per ledger case). So I changed the design to entrylog per ledger, which changed the logic of choosing entry log for the given ledger but the underlying changes of the abstraction of entry logger remained the same. This has been informed to the community formally in several community meet calls and Sijie in particular (multiple times) before proceeding. https://cwiki.apache.org/confluence/display/BOOKKEEPER/2017-06-01+Meeting+Notes https://cwiki.apache.org/confluence/display/BOOKKEEPER/2017-10-19+Meeting+Notes https://cwiki.apache.org/confluence/display/BOOKKEEPER/2017-11-30+Meeting+Notes https://cwiki.apache.org/confluence/display/BOOKKEEPER/2017-12-14+Meeting+Notes. As I requested in 2017/12/14 meeting I shared preview version of my code in community slack https://apachebookkeeper.slack.com/archives/C6G5104SF/p1513212259000262 and the code is shared in my GitHub repo - https://github.com/reddycharan/bookkeeper/tree/entrylogperledger . @eolivelli and @sijie were helpful to do early CR, provide some initial feedback and gave their approval. Finally as I envisioned, discussed and shared, after rebasing my changes on the recent community code I created formal pull request - https://github.com/apache/bookkeeper/pull/1201 and also updated this issue with formal description of entry log per ledger design which Iâ€™ve been discussing all along. Agreed, I could have updated this issue description with the new design back then itself. But Iâ€™m not sure if it would have made difference since this was communicated/explained multiple times in the community meets and particularly to the interested people. For the people who are coming across this work item for the first time, it is needed/required when I created the formal pull request and hence I made sure I updated issue with full description (new design) before sending the Pull Request. I can say community has changed a lot in the amount of activity, processes, systems used and the list of participants in the last year (2017). This Work Item has spanned during this transition phase (it took quite longer time, this is because of change in priorities on our side, me dealing with multiple repos - internals salesforce one and community, deluge of commits in the second half of 2017 and the need to rebase my commits on top of them every single time I wanted to push it forward) and I see why you might have different opinion, considering you werenâ€™t present in any of the above mentioned conversations/communications. But I see you are the one who migrated jira work item to git issues, but Iâ€™m not sure if it was automated task or well curated manual task. Hope this answers â€œwhen/where was this discussed partâ€ .
16. @reddycharan great summary. thank you!
17. **body:** @ivankelly @sijie @reddycharan Thnks a lot for this great conversation. While it is nice to discuss ASF requirements for a pull request and the level of design discussions on the community email, the real crux comes down to the technical merit of the patch. The last thing any of us wanted to do is to commit buggy and badly designed code. We must not have any durability issues, after all, we are the persistent layer of many services above us. If I can summarize @ivankelly's concerns: 1. This is a large patch and is difficult to review. @ivankelly while I agree that this is not a good practice, but for historical reasons, it ended up this way. We may not retroactively fix this unless @reddycharan thinks it is easy. So my request is to take some time and review this for now, and we can set up strict guidelines for future. Also, the community is lot active now (after Streamlio) before it was mostly silos. 2. It is a Highrisk patch as it touches the most sacred persistence layers. We must absolutely review/test to gain confidence in the patch. I request more eyes on this. Should add more tests and run with fault injection. Maybe an opportunity to vet test cases. 3. Abstraction is not in the right place. As @reddycharan and @sijie explained we went through many discussions around this. Charan even tried prototyping the abstraction at the ledger manager level. That ended up much more complicated than this. And some of the discussion was referred by email links given by @reddycharan Entry log per extent is very important for Salesforce for various reasons and I bet it will be essential to any use case which uses SSDs and doesn't have too many writable ledgers/ bookie at any given time. Moreover, it makes lot easy to implement storage tiering. We can have another meeting if needed, and my request is to move forward on the pure technicality of the patch rather than design preferences.
    label: code-design
18. **body:** I think a more confortable way to get this in is to split the patch into smaller changes. @reddycharan Just a question, is some code based on this change running in some production system ? @jvrao I will be a very happy user of this feature. Effectively it seems to me that this new kind of storage is vert different from InterleavedStorage. I remember we already discussed about this. Is it really a benefit to change InterleavedS and SortedS and not to start from scratch eventually creating a base class for common code?
    label: code-design
19. **body:** >Just a question, is some code based on this change running in some production system ? @jvrao No, not yet, we are in the process of pushing all of our internal changes to the community code and eventually use the community repo version, so that we can be close to the community. Since we are in this transition phase we kept hold of any big changes to our repo and instead work directly with the community

repo. >Effectively it seems to me that this new kind of storage is vert different from InterleavedStorage. I remember we already discussed about this. Is it really a benefit to change InterleavedS and SortedS and not to start from scratch eventually creating a base class for common code? Ideally with the design approach i chose, there shouldn't be any changes to InterleavedLedgerStorage class and SortedLedgerStorage class. Even now the changes made to these two classes are minimal. These changes are required for the following reasons - 1) Removal of unnecessary synchronization in InterleavedLedgerStorage methods. As described in http://mail-archives.apache.org/mod_mbox/bookkeeper-dev/201707.mbox/%3CCAO2yDyZ946fp2S_qR2iL178hPiXgrnFGb%3DpvkyK4ReSYAtNLBw%40mail.gmail.com%3E 2) minor changes to checkpoint logic, since in entryLogPerLedger it should not checkpoint when a new entrylog is created but instead it should checkpoint for every "flushInterval" period and also during this time it should flush both rotatedLogs and current active logs. 3) couple of minor changes in SortedLedgerStorage regarding how entryLogger methods are called and creation of flushexecutor and passing it to EntryMemTable in the case of entryLogPerLedger. The crux of the implementation change is in EntryLogger class and for the reasons explained above abstraction is done in EntryLogger class. >I think a more confortable way to get this in is to split the patch into smaller changes. @reddycharan Ok, I've spent multiple weeks on this work item. If it is going to make things easier on reviewers I can consider splitting this PR into multiple PRs (hoping it wont take heavy toll on me and complicate things further). Probably I can split like following **sub-task1** - Removal of unnecessary synchronization in InterleavedLedgerStorage methods. As described in http://mail-archives.apache.org/mod_mbox/bookkeeper-dev/201707.mbox/%3CCAO2yDyZ946fp2S_qR2iL178hPiXgrnFGb%3DpvkyK4ReSYAtNLBw%40mail.gmail.com%3E **sub-task2** - move the complete logic of flushIntervalInBytes from EntryLogger to BufferedChannel **sub-task3** - make changes to SyncThread/checkpoint logic, so that incase of entrylogperledger, checkpoint happens for every flushInterval but not when active entrylog is created/rolled over. **sub-task4** - introduce parallel (have just one Runnable per ledger) EntryMemTable flusher for SortedLedgerStorage which can be used in the case of entrylogperledger **sub-task5** - Introduce EntryLogManager abstraction in EntryLogger and let it deal with the singleentrylog/entrylogperledger functionality and core business logic. As you can imagine, sub-task5 is going to be the crux of this work item and majority of my code change and the design choice of abstraction at EntryLogger is going to remain.
    label: code-design
20. the split LGTM
21. Okay for the split for me
22. Hey @charan, @jvrao, @sijie > to answer â€œwhen/where was this discussed partâ€ , I can point you to the meeting notes, mail exchanges, slack messages and git links which I shared with the community throughout the period. The only mails I can see for this are the two linked threads, and the subject of these have very little to do with per ledger storage. It seems most of the discussion took place on other channels. The meeting notes show it was discussed but not what was discussed. Participation in the meeting is dependent on being available at that time, and no detailed record is retained. Slack messages depend on everyone being online at that time. Synchronous communication makes it hard for people to participant. This is why ASF encourages that all discussion happens on the mailing lists [1][2]. The reason I'm taking exception to this, is because when I proposed that this be done a different way, I was told: - "Sijie, JV and our internal team had long conversation about the same and I responded to community aswell in one of the mail" [3] - "I think this has been discussed a while back. I would suggest no holding this for requesting a new ledger storage implementation" [4] - "I am suggesting here is to respect to what have been discussed and agreed on". [5] Suggesting that the design of the feature is a settled issue, and a long open discussion had already occurred for it. This is not the case. â€œIf it didnâ€™t happen on-list, it didnâ€™t happen.â€ [2]. > While it is nice to discuss ASF requirements for a pull request and the level of design discussions on the community email, the real crux comes down to the technical merit of the patch. The ASF's motto is "Community over code". > We can have another meeting if needed, and my request is to move forward on the pure technicality of the patch rather than design preferences. Meetings are the problem here. The discussion should be on list, where everyone can see it and there's a permanent record. I'll respond to the technical/patch size issue in a different comment. [1] http://www.apache.org/foundation/how-it-works.html#communication [2] http://theapacheway.com/community/ "Mailing list" section [3] https://github.com/apache/bookkeeper/issues/570#issuecomment-368597767 [4] https://github.com/apache/bookkeeper/issues/570#issuecomment-368654064 [5] https://github.com/apache/bookkeeper/issues/570#issuecomment-368826780 [6] http://www.apache.org/dev/contributors#comdev
23. **body:** @reddycharan The split looks better than what's there, but I still think sub-task4 is unnecessary and sub-task5 is at wrong level. Another option, which could be less work given what you have already, is to duplicate the classes that you are modifying, and making the changes you need there. This way you don't need to touch SortedLedgerStorage, you don't modify code that all other ledger storages use, and the patch should just be whole file additions, and a small modification for enabling users to use the new ledger storage. It would mean code duplication, but it's easier to factor out duplicate code than to disentangle code. Also, the risk of submission would be zero. To be clear, I appreciate that a lot of work has gone into this, but this doesn't trump my other concerns. Also, per ledger ledger storage is something that I'll be very happy to see in-tree. I even did an implementation for benchmarking years ago[1]. Only ever ran on HDD, so performance sucked. [1] https://github.com/ivankelly/bookkeeper/tree/PerFileLedgerStorage
    label: code-design
24. > It would mean code duplication, but it's easier to factor out duplicate code than to disentangle code. Its a no-no for me; Not just this issue, For any feature, in any context - code duplication never scales, hard to manage and error/bug prone. > Meetings are the problem here. The discussion should be on list, where everyone can see it and there's a permanent record. As I hinted in my response when this happened community was more or less organized as silos. It was mostly Salesforce, Sijie, Matteo. Enrico was just becoming active. So if there was a discussion among JV, Charan, Sijie, Matteo it was the whole community at that time. Charan shared the code couple of times on mailing list and on slack, though it's not in a doc form. Again my request is to look forward than pointing fingers. Let's agree on how to move forward. @reddycharan is there any way you can sub-divide your subtask-5?
25. **body:** > Synchronous communication makes it hard for people to participant. This is why ASF encourages that all discussion happens on the mailing lists [1][2]. > Meetings are the problem here. I disagree with that. Meetings are sometimes better than discussion on the list. We just need to make sure us doing better job on summary and keeping the records onwards. The community has been moving much faster because of meetings which provide more effective ways on communication and discussions. > This is not the case. â€œIf it didnâ€™t happen on-list, it didnâ€™t happen.â€ [2]. The community was inactive until recently. Most of the discussions were happening between Twitter, Yahoo and Salesforce through direct meetings, where the meetings were not even public to other people to attend (where the meetings were evolved into the community meetings now). We probably did badly at old days, but that's how we moved the community forward and iterate the community into its current form. It is much better than before. We are following a better apache process on changes happen in bookkeeper. The discussions done before probably didn't follow apache way very well. But it doesn't mean they don't exist. That's why I suggested respecting to the efforts that people have done in the past. The community is still small, the people are still the same people who were involved in the discussions. It makes no sense pointing ASF policies around to rule out those discussions made before we formed a better community process. We all know we need to move forward with a better process. So I would hope us stop any comments like this. It is not helpful at all. Let's focus on technical discussions and let the community decide what is the approach we should take here, take it and move forward. ---------- Technical parts > sub-task5 is at wrong level. > Another option, which could be less work given what you have already, is to duplicate the classes that you are modifying, and making the changes you need there. This way you don't need to touch SortedLedgerStorage, you don't modify code that all other ledger storages use, and the patch should just be whole file additions, and a small modification for enabling users to use the new ledger storage. > It would mean code duplication, but it's easier to factor out duplicate code

than to disentangle code. Also, the risk of submission would be zero. A no from me as well on code duplication. still think EntryLogManager is a good approach to take here for multiple entrylogs.
label: code-design

26. As far as I understood, EntryLogger is the abstraction layer for all of the operations on entrylog files. Be it how entries are written to entrylog file (how active/rotated entrylog files are organized), read from entrylog files (how read files are organized), flush/checkpoint path and getting info about the state of entrylog files for garbage collection/compaction. Also, I tried to explain in detail why it is the right thing to do abstraction in EntryLogger in this comment https://github.com/apache/bookkeeper/issues/570#issuecomment-368597767 Entrylogger is the right place to deal with multiple entrylogs, it is more organic, less churn of the code, since ideally other components of the bookie doesn't need to be modified for anything to do with entry log files. But yes, regarding sub-task4, it is valid point to raise the need of SortedLedgerStorage if we are going to have entrylog per ledger. I need to have some perf numbers to validate the write/read scenarios in the case of entrylog per ledger using InterleavedLedgerStorage vs SortedLedgerStorage. **sub-task4** - introduce parallel (have just one Runnable per ledger) EntryMemTable flusher for SortedLedgerStorage which can be used in the case of entrylogperledger > @reddycharan is there any way you can sub-divide your subtask-5? Yes, I think it should be possible. Probably in the first task I could introduce EntryLogManager interface and EntryLogManagerForSingleEntryLog and in the final task I can introduce EntryLogManagerForEntryLogPerLedger. So **sub-task1** - Removal of unnecessary synchronization in InterleavedLedgerStorage methods. As described in http://mail-archives.apache.org/mod_mbox/bookkeeper-dev/201707.mbox/%3CCAO2yDyZ946fp2S_qR2iL178hPiXgrnFGb%3DpvkyK4ReSYAtNLBw%40mail.gmail.com%3E **sub-task2** - move the complete logic of flushIntervalInBytes from EntryLogger to BufferedChannel **sub-task3** - make changes to SyncThread/checkpoint logic, so that incase of entrylogperledger, checkpoint happens for every flushInterval but not when active entrylog is created/rolled over. **sub-task4** - introduce parallel (have just one Runnable per ledger) EntryMemTable flusher for SortedLedgerStorage which can be used in the case of entrylogperledger. (evaluate the need of SortedLedgerStorage by doing perf comparisons) **sub-task5** - introduce EntryLogManager interface and EntryLogManagerForSingleEntryLog **sub-task6** - introduce EntryLogManagerForEntryLogPerLedger Thanks guys for providing the feedback. Will proceed with the plan and start creating new pull requests for the sub-tasks. But I'll leave the existing pullrequest (https://github.com/apache/bookkeeper/pull/1201), since it might be helpful to refer to get the full picture.
27. @reddycharan thank you so much for the response. the plan looks good to me. +1 to proceed.
28. > It makes no sense pointing ASF policies around to rule out those discussions made before we formed a better community process. Similarly, these discussions shouldn't rule out further discussion when it is opened up to the wider community. In-person and 1-to-1 communication is always going to happen, but when they do, anything decided will still have to be justified when brought to the community as a whole. Anyhow, I've made the point I wanted to make here, and hopefully we can avoid the same kind of problems again in the future. I'm drawing a line under this now, will respond to technical stuff in another comment. ---
29. **body:** @reddycharan I agree that EntryLogger is the correct place for this abstraction. However, I don't think EntryLogger itself should decide whether it uses a log per ledger or not. This affects other aspects of the LedgerStorage, the storage should own this decision. I would suggest constructing the EntryLogManager at the ledger storage, and passing this as a parameter to the entrylogger. This will make EntryLogger easier to test, as the manager can be mocked. It also means that SortedLedgerStorage can be removed from the picture, as we can force SortedLedgerStorage to always construct the interleaved EntryLogManager. The parallel memtable flush change may still be useful, but it shouldn't be a concern of this change set. If we want it, we should pull it in separately.
label: code-design
30. This is the master issue I have reopened it

github_pulls:

github_pulls_comments:

github_pulls_reviews:

jira_issues:

1. **summary:** Multiple active entrylogs
description: Current bookkeeper is tuned for rotational HDDs. It has one active entrylog, and all the ledger/entries go to the same entrylog until it is rotated out. This is perfect for HDDs as seeks and moving head allover the disk platter is very expensive. But this is very inefficient for SSDs, as each SSD can handle multiple parallel writers, also this method is extremely inefficient for compaction as it causes write amplification and inefficient disk space usage. Our proposal is to have multiple active entrylogs and a configuration param on how many parallel entrylogs the system can have. This way one can have ability to configure to have less (may be one) ledger per entrylog.

jira_issues_comments:

1. {quote} But this is very inefficient for HDDs {quote} [~jujjuri] did you mean SSD ?
2. Yes I mean SSD. Corrected. Thanks.
3. **body:** Issue: In Bookie's EntryLogger, we are having only one current active entryLog and all the ledger/entries go to the same entrylog. This is perfect for HDDs as file syncs, seeks and moving head allover the disk platter is very expensive. But having single active Entry Log is inefficient for SSDs, as each SSD can handle multiple parallel writers. Also, having single active EntryLog (irrespective of LedgerStorage type - interleaved/sorted), is inefficient for compaction, since entries of multiple ledgers will end up in an entrylog. Also in SortedLedgerStorage , in the addEntry request we flush EntryMemtable, if it reaches the sizelimit. Because of this we are observing unpredictable tail latency for addEntry request. When EntryMemTable snapshot of size (64 MB) is flushed all at once, this may affect the journal addentry latency. Also, if the rate of new add requests surpasses the rate at which the EntryMemTable's previous snapshot is flushed, then at a point the current EntryMemTable map will reach the limit and since the previous snapshot flush is in progress, EntryMemTable will throttle new addRequests, which would affect addEntry latency. Goals: The main purpose of this feature is to have efficient Garbagecollection story by minimizing the amount of compactions required and the ability to reclaim the deleted ledger's space quicker. Also with this feature we can lay foreground for switching to InterleavedLedgerStorage from SortedLedgerStorage and get predictable tail latency. Proposal: So proposal here is to have multiple active entrylogs. Which will help with compaction performance and make SortedLedgerStorage redundant. Design Overview: - is to have server configuration specifying number of active entry logs per ledgerdir. - for backward compatibility (for existing behaviour) that config can be set to 0. - round-robin method will be used for choosing the active entry log for the current ledger in EntryLogger.addEntry method - if the total number of active entrylogs is more than or equal to number of active ledgers, then we get almost exclusivity - For implementing Round-Robin approach, we need to maintain state information of mapping of ledgerId to SlotId - there will be numberofledgerdirs*numberofactiveentrylogsperledgerdir slots. a slot is mapped to ledgerdir, but the activeentrylog of that slot will be rotated when it reaches the capacity. - By knowing the SlotId we can get the corresponding entryLogId

associated to that slot. - If there is no entry for current ledger in the map, then we pick the next in order slot and add the mapping entry to the map. - Since Bookie won't be informed about the writeclose of the ledger, there is no easy way to know when to remove the mapping entry from the map. Considering it is just <long ledgerid, int slotid> mapentry, we may compromise on evicting policy. We can just use some Cache, which has eviction policy, timebased on last access - If a ledgerdir becomes full, then all the slots having entrylogs in that ledgerdir, should become inactive. The existing mappings, mappings of active ledgers to these slots (active entrylogs), should be updated to available active slots. - when ledgerdir becomes writable again, then the slots which were inactive should be made active and become eligible for round-robin distribution - For this feature I need to make changes to checkpoint logic. Currently with BOOKKEEPER-564 change, we are scheduling checkpoint only when current entrylog file is rotated. So we dont call 'flushCurrentLog' when we checkpoint. But for this feature, since there are going to be multiple active entrylogs, scheduling checkpoint when entrylog file is rotated, is not an option. So I need to call flushCurrentLogs when checkpoint is made for every 'flushinterval' period
   label: code-design
4. **body:** [~jujjuri] [~reddycharan18@gmail.com] This sound very interesting. Now I can see clearly way JV wrote on the mailing list that maybe clients could send some hint to the bookies that a ledger has been gracefully deleted/closed
   label: code-design
5. [~sijie@apache.org] created writeup for this work item and discussed about it in last community call (May 18th)
6. Migrated to github https://github.com/apache/bookkeeper/issues/570