Item 302
git_comments:

1. tmaster -> ckptmgr messages
2. This is the message that a stmgr sends when it wants to register itself with the checkpoint manager
3. This is the message that the stmgr sends to its local tasks to begin initiating checkpoint
4. Message sent to stmgrs by the tmaster to initiate checkpointing
5. Message sent by tmaster to stmgr asking them to reset their instances to this checkpoint
6. This is the message that a tmaster sends when it wants to register itself with checkpoint manager
7. Message sent by stmgr to tmaster informing it about the fact that we stored a checkpoint belonging to the instance
8. This is the response that Checkpoint Mgr sends to StMgr for its get instance state request
9. tmaster -> stmgr messages
10. Message sent by stmgr to Instance to start processing This is related to StartStmgrStatefulProcessing message above but between stmgr -> instance.
11. Message sent by tmaster to stmgr to start processing For stateful processing, all the topology components should be rolled back to a consistent state before they can start processing.
12. Any checkpoints older than this can be cleaned
13. This is the request that StMgr sends to Checkpoint Mgr to fetch the instance state data
14. Message sent by ckptmgr to tmaster about the cleanup request
15. This is the message that checkpoint manager sends when it receives the register request from tmaster
16. This proto encapsulates the info associated with state of an instance
17. One can add more meta data about this ckpt later
18. Information about the instance whose state this is
19. This is the message that checkpoint manager sends when it receives the register request from stmgr
20. message stored in the state manager by the tmaster
21. Message sent by tmaster to ckptmgr to cleanup old checkpoint state.
22. An ordered list of the globally consistent checkpoints that have been snapshotted and can be recovered from. 0th element is the most recent one, 1st is the next recent and so on
23. This is the message that the instance sends when it wants stmgr to store its state
24. stmgr -> Instance messages
25. The messages in this file are for stateful processing and providing exactly once semantics. Below are the various sequence of events that happen in a few(non exhaustive) scenarios On Startup For topologies needing exact once, the startup sequence of tmaster/stmgr/instance is a little different. Normally, upon startup, all stmgrs connect to tmaster and then tmaster computes pplan and distributes to stmgrs, which in turn send it to all their local instances. As soon as instances get their pplan, they can start processing(spouts emitting tuples, etc). However for stateful topologies, this is a little different. First in addition to the pplan, tmaster needs to send the last consistent checkpoint id(if any) to recover from as well. Secondly instances need to recover their prev state before they can process any tuples. WRT messages below, the startup sequence is the following. From the StatefulConsistentCheckpoints message stored in the state manager, Tmaster picks the latest checkpoint_id and sends RestoreTopologyStateRequest to all stmgrs. Each stmgr does a GetInstanceStateRequest/GetInstanceStateResponse dance with its local checkpoint manager for all of its local instances to fetch the state. Upon fetch, stmgr does a RestoreInstanceStateRequest/RestoreInstanceStateResponse dance with the corresponding instances to restore the state. Completion of which, stmgr sends out the RestoreTopologyStateResponse message to the tmaster. Once the tmaster receives this from all stmgrs, it sends out the StartStmgrStatefulProcessing message to all stmgrs which inturn send out StartInstanceStatefulProcessing to all their local instances. This gets the topology humming Periodic checkpoints Every so often(as dictated by TOPOLOGY_STATEFUL_CHECKPOINT_INTERVAL) tmaster sends out StartStatefulCheckpoint message to all stmgrs. Each stmgr sends out InitiateStatefulCheckpoint message to each of its local spout instances. Because of the best effort nature, this is delivered as a message. After instance does its checkpoint, it sends StoreInstanceStateCheckpoint to stmgr to store its checkpoint. Stmgr passes this to ckptmgr via SaveInstanceStateRequest/SaveInstanceStateResponse dance with its local checkpoint manager. Upon getting a successful SaveInstanceStateResponse message, stmgr sends out a InstanceStateStored message notifying tmaster that the state of a particular instance has been saved. Parallelly stmgr also sends out DownstreamStatefulCheckpoint message to each of the downstream components to do the distributed Lamport style checkpointing. Meanwhile at tmaster, when it receives

InstanceStateStored from all instances for a particular checkpoint_id, it has reached globally consistent checkpoint and it stores it in the state manager as StatefulConsistentCheckpoint Failure Recovery Either when stmgr dies, or when an instance dies(which is notified by the stmgr using the ResetTopologyState message), tmaster initiates the recovery mechanism. The recovery mechanism is exactly the same as the initial startup described above
26. This is the message that instance sends to its stmgr when done with restoring the state
27. This is the message that stmgr sends to its instance asking them to restore their state
28. Every Attempt made by Tmaster to restore to a globally consistent checkpoint is tagged with a unique id. The stmgrs include this in their RestoreTopologyStateResponse below. This way tmaster could try restoring to the same checkpoint_id multiple times and still keep track of for which attempt the restore was in the responses
29. Message representing information about a globally consistent checkpoint
30. After successfully saving state, this is what Checkpoint Mgr sends back as response
31. Message sent by stmgr to tmaster asking it to reset the topology to some valid checkpoint. This is sent either if stmgr dies and comes back up or if an instance dies.
32. This is the request that StMgr sends to Checkpoint Mgr to store instance checkpoint data
33. Once stmgr receives StoreInstanceStateCheckpoint from an instance, it sends this message to all of that instance's downstream components' stmgrs to propagate the checkpoint marker
34. Message that stmgr sends to tmaster after it restores all of its local instances to a checkpoint_id
35. stmgr -> stmgr messages
36. stmgr -> ckptmgr messages

git_commits:

1. **summary:** Added proto files declaring all the messages transmitted across (#1829)
   message: Added proto files declaring all the messages transmitted across (#1829) * Added proto files declaring all the messages transmitted across all different heron components for stateful processing and exactly once semantics * Made changes to the proto structures based on feedback * Seperated out inner message to top * Added description of all the messages. Changed field names as per feedback * Space before { * Fixed comments * Fixed comments * fixed comments

github_issues:

github_issues_comments:

github_pulls:

1. **title:** Added proto files declaring all the messages transmitted across
   body: all different heron components for stateful processing and exactly once semantics

github_pulls_comments:

1. LGTM

github_pulls_reviews:

1. what is this txid for?
2. when is this message sent? Is it only sent once during the topology startup? This message looks strange to me that if a topology is configured with stateful processing, then once it starts running, the processing is stateful already.
3. do we need the ability to clean only one particular checkpoint?
4. also response with the cleaned checkpoint ids would be better?
5. s/stmgr/stmgr_id
6. add `required heron.proto.system.Status status`?
7. the same question as `StartStmgrStatefulProcessing `
8. is this message to handle the case where a downstream instance has multiple updtream instances?
9. Instead of storing an string id and a string[] backupId, could we instead store an ordered array of Checkpoint objects, most recent first? Using an object instead of a string would allow for extensibility to bind more metadata to a checkpoint than just id (e.g., creation time, location). We could adjust how many

checkpoints we keep via config (either time based or absolute count), which becomes the size of the array.
10. message is stored in state manager right? zk is just one of state managers impls.
11. Added more comments to explain this. Let me know if it makes more sense now :)
12. Added comments.
13. This is just the oldest checkpoint that needs to be preserved. All older ones can be cleaned
14. added
15. done
16. added
17. Explained earlier
18. This is the marker message. Added comments to explain
19. Made changes
20. changed
21. Why an inner class? If we make this a top level class it can be used on its own without the collection, like in the RPC for example while working with a specific checkpoint.
22. Done
23. If this is an RPC request object should it be `StartStatefulCheckpointRequest`?
24. Is this the payload of an RPC request? If so it should follow the same standard and be `ResetTopologyStateRequest`.
25. We do follow that terminology where all request/responses are labelled as such. Anything else that is not a request/response are just best effort messages that go from one component to another. For some, a response is not expected. For others it cannot be a request because in Heron c++ client/server communication, only clients can send a request and a server cannot.
26. Same as above
27. Do the heron client/server model use a protobuf RPC interface or do they just send protobuf object bytes on raw sockets?
28. the latter(i.e. use protobuf object bytes)
29. Why does the tmaster need to be aware of specific instance state? Why not have the stream manager manage the states of the instances under it's control?
30. When/how is this message used and how does the stream manager get into the state that is before or after this one (i.e., one where it's not started)?
31. Why does TMaster register itself with the checkpoint manager instead of the other way around, like the way state manager registers with TMaster? TMaster is already discoverable in StateManager. Also space before {
32. Why would checkpoint manager need to be told the topology name and id? This info is static and can be passed to checkpoint manager at startup. Same question for `RegisterStMgrRequest` below.
33. If these fields are both optional neither of them can be set, which I assume would be invalid. If that's the case can we enforce that one of them is set view `oneof`? That would assure that every possible field setting is valid. https://developers.google.com/protocol-buffers/docs/proto#oneof What's the expected behavior when `oldest_checkpoint_preserved` is unset and `clean_all_checkpoints=false`?
34. `cleaned_checkpoint_ids`
35. "establish the connection" should be "register itself" no?
36. space before {
37. Is this the message or is this the state info that's part of another message? Reading above it seems the latter but the description states the former. If it is the former, can we put a verb in this object, to make it more clear what action is being requested?
38. typo: instace
39. Same question as above w.r.t. state changes. If this request triggers processing to start, how does the instance get into the state where it's not processing?
40. I thought checkpointing was non-blocking, but this comment sounds like checkpointing happens serially from one component to the next. Why? Also why does each instance broadcast this message to all instances of the downstream component?
41. Can we more clearly specify state than an arbitrary array of bytes? What is this byte array and how does the StateStore implementor know how to read the bytes? How for example would I persist the byte array in a key value store in a way where I could inspect individual records?
42. We need to associate the checkpoint with a packing plan or physical plan so we know it's valid to reuse upon topology restart.
43. This is purely for information sake only. Mostly used for printing/debugging

44. The idea is to use ckptmgr as some sort of service by the tmaster. TMaster upon startup would connect to it(which is also running in the same container) and register itself
45. This is for the sake of consistency wrt all other register requests. The original idea was sanity checking
46. Current phase only treats it as arbitrary bytes. The aim is to get the flow mechanism correct, before expanding this
47. distributed checkpointing is non-blocking. However the mechanism of how they are triggered are serialized. This is part of the Lamport's algorithm. To be part of a consistent checkpoint, an upstreamer needs to checkpoint first and then send these markers to all potential downstream components. Downstream components can checkpoint only after they received all their upstream components.
48. oneof is not supported by the protobuf version that we are using
49. Rather than do this in the design doc(which is seperate entity) I would rather include some sort of flow in the file itself
50. This is done. Please take a look
51. Added more in the header section
52. Done style fix
53. done
54. changed
55. fixed
56. Fixed this
57. Fixed
58. Why is this at the instance level and not the component level? Couldn't "Once an instance is done checkpointing" be "Once a component is done checkpointing"? Also, if each instance is broadcasting to all of it's downstream instances, that implies that the downstream instances need to be aware of and track the id set of all of its' upstream instances, and which ones have sent the message? Pushing this responsibility onto each instance seems unnecessary. Instead could TMaster message all instances of Component B (via stream managers) that it's time to checkpoint, based on it's knowledge that all Component A instances are complete?
59. This text description is helpful, but it would still be great to have a sequence diagram of these flows. Visualizing the interactions between the many agents really helps to understand the system and to expose quirks. Here's an example that we created to clearly illustrate the Topology Submit Sequence (which is less complicated than these flow IMO), for example: https://twitter.github.io/heron/docs/concepts/architecture/
60. upon startup
61. all stmgrs connect to
62. in turn
63. s/exactly once semantics/stateful topologies/
64. "because instances are all busy recovering their state" assumes that the reader knows the startup sequence of events, which is what we're trying to describe. Could you list the sequence of startup events for stateful topologies? It should be very clear to the reader the order of things like fetching the packing plan, initializing state, registering with stream managers, physical plan generation, etc.
65. receives
66. Couple of clarifications:- 1. Just to be clear in Heron parlance component means one logical component(like spout/bolt). Instance is one running process of it. Thus there are 5 instances of 'wordSpout' component. In this case, it is instance level. 2. Chandy Lamport's algorithm necessitates that downstream instances are aware of all possible upstreamers. In practice this is performed at the stmgr so the real instance code is not aware of it. We cannot involve tmaster here because then a) it is not decentralized and b) only local state inside stmgr is sufficient
67. Two questions: 1. Could deactivate/activate be used for this, or would that overload the activation semantics? 2. This message would be sent when recovering to a given checkpoint, during which processing is already started, so this message would also be used to reset a started topology to a given checkpoint, correct?
68. s/exactly once semantics/stateful processing/
69. 1. activate/deactivate are meant for totally different things and imo not applicable. 2. yup. In future we could add some commands like that to the cli itself
70. fixed
71. fixed
72. reworded
73. fixed

74. The instance itself doesn't need to be aware of the upstream/downstream task infromations, stmgr can manage this and decide when to ask a local instance to do the checkpoint by sending `message InitiateStatefulCheckpoint` I suppose.
75. omit: sends
76. removed
77. I think your comment was about my request for a sequence diagram. What about my comment about how we need to bind a checkpoint to a given packing plan or physical plan to know if it's valid upon startup?
78. It's listed in the version 2 spec: https://developers.google.com/protocol-buffers/docs/reference/proto2-spec What's the minimum version that supports it? We should consider upgrading, since this was included in 2.x.
79. Can you add a TODO next to that then to clarify that we don't want to ship with this being a byte array?

jira_issues:

jira_issues_comments: