Updating schedule-generator with deltas

Spike: Schedule-generator Deltas

- **Conducted by:** Fabrizio Colucci & Oli Bowker (with exec. consultants Ray Cheung & Karl Lloyd)
- **Backlog Work Item:** IPLAYERTVV1-14412

Goals

1. Validate proposed algorithms for dealing with service-schedule and catalogue updates and deletes flowing into the schedule-generator - no optimisations at this point
2. Outline what updates would be required to the existing cold-start mode to be compatible with this
3. Suggest what potential optimisations could be made, taking into consideration the current typical delta throughput from both sources

Method

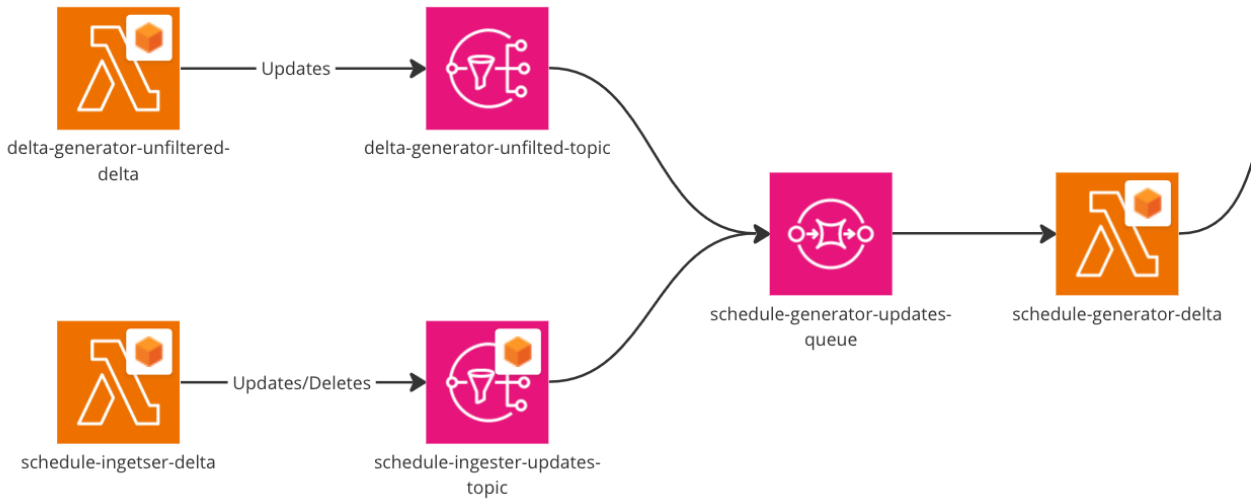
1. Build prototype to Int environment.
2. Use unit tests to cover list of scenarios in the above parent ticket.
3. Promote spike to Test environment to analyse capability of handling throughput of real data.
   a. Add a delta-generator outgoing SNS for notifications. Send only on Int & Test?
   b. Other infrastructure exists in schedule-generator spike branch - attach incoming SQS to existing schedule-ingester SNS, etc. Deploy this in parallel with current schedule-generator.
4. Analyse the complexity/maintainability of solution.

Test scenarios

See IPLAYERTVV1-13966 description for initial list of scenarios to cover.

Evidence

The repo and unit tests for algorithm and subsequent AWS architecture. The delta lambda can be found here and the logs for the lambda can be found here.



New architecture for the schedule-generator

Metrics have been setup in grafana to show throughput on TEST, under the `Schedule Generator (SPIKE work)` tab.

- Metric to track number of updates and deletes per type (episode/series/brand)
- Metric to track number of ignored updates and delete per type (episode/ancestor)
- Metric to track general lambda errors
- Metric tracking average run time, invocations and oldest message in queue.

Conclusions

Approach

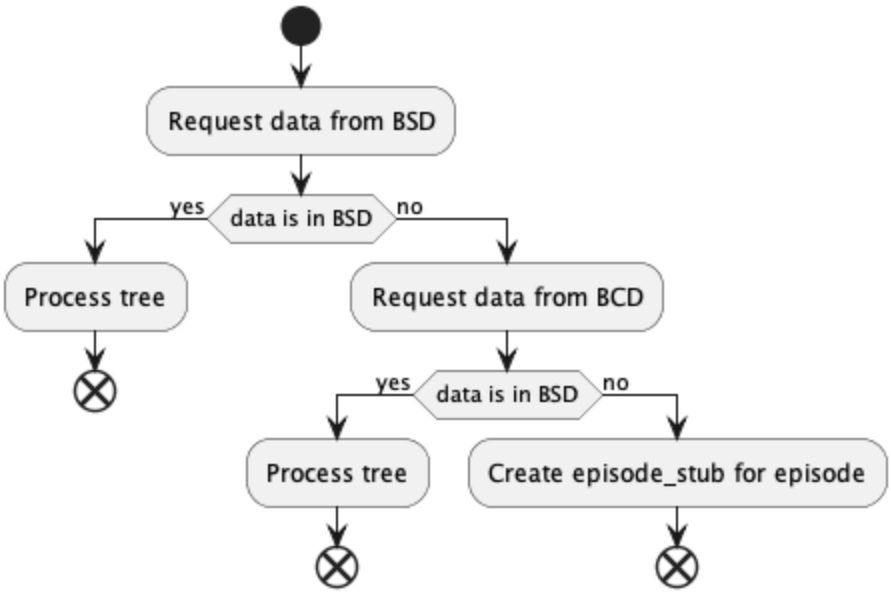
The final approach consisted of repositories requiring change.

- A spike repo was created that housed the new generator code. This code writes to the redis keyspace *{bsd_v2.0_spike}* and *{bsd_v2.0_ref_data_spike}*. Unit tests were written that covered scenarios specified in the intial breif.
- Updates to the delta generator were added to add an SNS topic on the unfiltered lambda, which posts **UPDATES** events only to the schedule generator SQS.

On schedule **UPDATE** all broadcasts are iterated through and respective data episode/series/brand data is persisted into the *{bsd_v2.0_spike}* keyspace by the below algorithm. With broadcaast lists of episodes being populated with an array of

`*sid|broadcast_pid*` identifiers. This list is used to know what schedules need updating when episode/series/brand updates come in.

- Edge case - broadcast_pid or episode is changed in an existing schedule. Remove old identifier from the old referenced episode, if that broadcast_list is empty then remove the episode.



Schedule update tree building logic.

On schedule **DELETE** all broadcast are iterated through, with associated `*sid|broadcast_pid*` identifiers being removed from the episodes broadcast_list. If this list ends up being empty, the episode is removed from *{bsd_v2.0_spike}* alongside the schedule.

On episode **UPDATE** get the new episode data from *{bcd_v2.0}* and then append the broadcast_list and new seq.

On series/brand **UPDATE** get the new series/brand from *{bcd_v2.0}* and then append a new seq number to it. Then determine the linked episodes in *{bsd_v2.0_spike}* and update the relevant schedules in their broadcast_lists.

Catalogue **DELETES** are not explicitly handled.

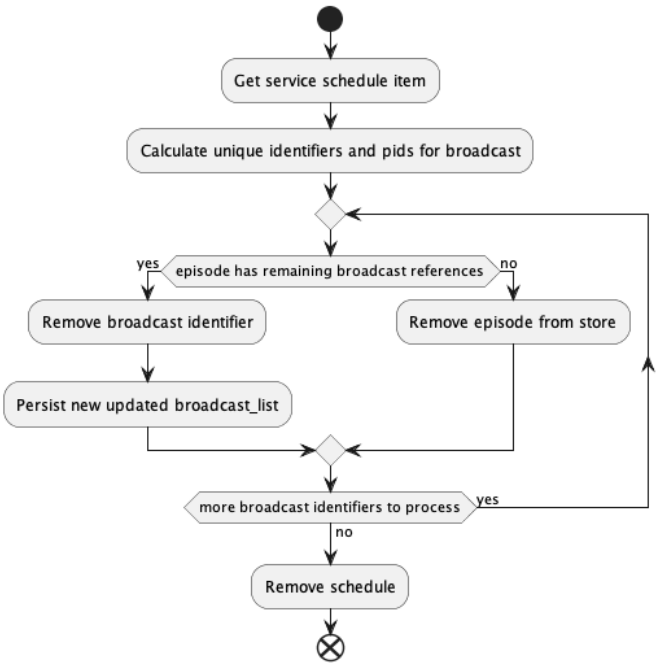
Coldstart

Coldstart functionality was not fully implemented during the spike however below is a list of things that would need to be handled.

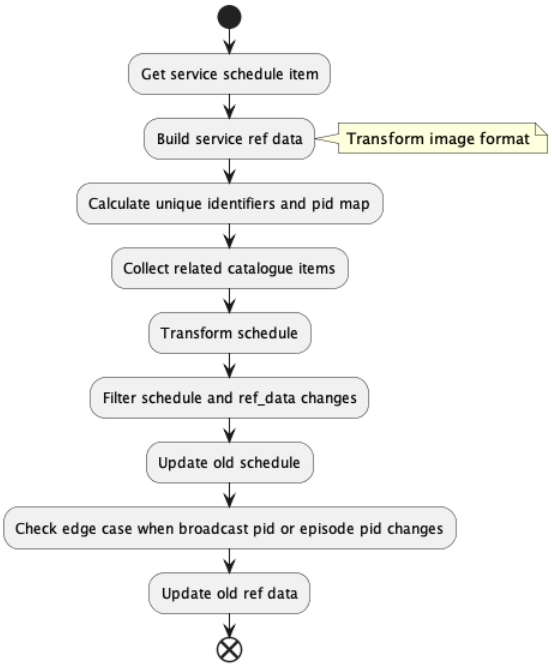
- Standard *'cut-off' f*or the lamdba SQS mappings would need to be added for the SQS going into the schedule generator. This also includes perging the queue.
- Protection when the delta-generator coldstarts. Implement some form of threshold of amont of updates that can be sent to the schedule-generator at once. If this threshold is breached, coldstart the schedule generator as we do in the aggricat.
- Purge all catalogue objects from *{bsd_v2.0}* and compile episodes/series/brands.

Garbage Collector

The current system only handles the deletion of episodes and schedules from the *{bsd_v2.0_spike}* keyspace. It was determined a single job ran daily to clean up the series and brands would be better as all data from the *{bsd_v2.0}* keyspace would have to be read and parsed for every delete. Considering deletes only happen once a day, we may as well tidy up once a day.



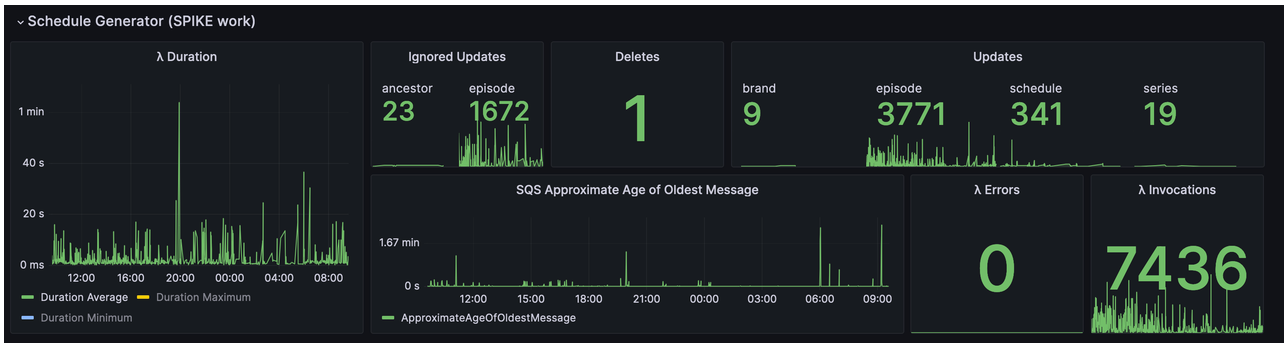
Activity for schedule delete.



Activity for schedule update.

Monitoring

Below is moniroing stats for a 24 hour period. It's important to note the ignored updates, actual updates and deletes will all be higher when we do this for real. This is due to the spike not coldstarting, therefore there are less schedules in which in the store. However this does nicely show a *maximum* amount of events we're likely to recieve.



Pros	Cons
Schedules can somewhat rely on it's own data store after initial setup of data is done.	**Large duplication of redis data across {bsd_v2.0} and {bcd_v2.0}.**
New code and infrastructure is all things that we have done before, nothing new to learn.	**Hard to parallelise if we choose to.**
Changesets should be easier to implement using a solution like this.	**Catalogue and Schedule pipelines are not separate, (however this is already how it works on LIVE).**

Potential Improvements

- Schedule-generator
  - Parrelise schedule updates triggered by episode/series/brand changes. An episode change then triggers updates for any schedule that it refers to, brand and series also do this but for the entire tree. An episode/series/brand present in BBC One can therefore will spin off a schedule change for all regional and hd variants. If a brand like eastenders was to change for whatever reason, this would result in 100s of schedule changes. This is a must.
  - Remove all catalogue deletes from schedule generator, and tidy up anything with an empty broadcast_list, and it's parents in the garbage collector. This would simplify the current schedule-generator code and make garbage collection easier.
  - Implement episode→broadcast mappings in DynamoDB instead of having broadcast_list in episodes. This would remove the need for redis duplication and would allow us to parallelise using optimistic locking in the future if we choose/need to.

        - This could be done at the ingester, once more simplifying the schedule-generator even more.
- Delta-generator
    - More filtering could be applied as we only care if titling, synopses, subtitles, (images?) or warning_text has been updated. If a dynamoDB mapping approach is taken, then a lookup could be done and only relevant pids would be sent on top of this (maybe over engineering).
    - Make code more explicit that it wants to publish to SNS, don't rely on their not being a manifest key and the version not being v1.1.

Next Steps

If we are happy to continue with the above approach, Oli and Fab will create/slice tickets and a kick off can be planned.