Dynamic Resource Allocation for Structured streaming

Author: Pavan Kotikalapudi

Shepherd: TBD

Ticket: SPARK-24815

Q1. What are you trying to do? Articulate your objectives using absolutely no jargon.

Provide Spark structured streaming with the ability to dynamically scale resources based on the heuristics of trigger interval.

Q2. What problem is this proposal NOT designed to solve?

This solution is not meant to be used in conjunction with traditional DRA configs to run spark batch jobs.

Q3. How is it done today, and what are the limits of current practice?

Currently there is no implementation of dynamic resource allocation of spark cluster resources for structured streaming in spark. Spark provides dynamic allocation scheduling to adjust resources based on the workload. But Spark's dynamic allocation is typically suitable to address use cases of batch mode (i.e jobs which run at scheduled times), it doesn't scale out/back well for streaming cases.

If we use batch based DRA for streaming workloads It will scale out when there are spark tasks queued in a particular spark stage of the job and scale back when the executors are idle without any running tasks, that sounds alright right?

Not really, We have 2 issues here:

1. Scale out policy is based on `spark.dynamicAllocation.schedulerBacklogTimeout` config and it scales out if a job is running in a particular stage for more than time mentioned in the config above. That would be a problem if multiple stages of a job are all small enough that it wouldn't cross schedulerBacklogTimeout but the overall runtime of the job (i.e micro-batch) crosses the scale-out threshold.

2. The scale back policy is by idleness of an executor. If we set executor idle timeout below trigger interval, it will constantly decommission and upon new micro-batch scales out executors every trigger interval. That will cause additional executor bootstrapping delays as we have streaming micro-batches which trigger every few mins/secs. This will hurt the flow especially in stateful streaming jobs where you have to transfer state in and out to checkpointing/decommissioning storage for each micro-batch. To avoid this If we set the executor idle timeout above trigger interval, the executors never become idle as they are used by spark Job for max parallelism based on Source.

Q4. What is new in your approach and why do you think it will be successful?

The new approach is to enable DRA for Structured Streaming apps based on the heuristics of Trigger interval.
The trigger interval will help us determine the scale-out and scale-back thresholds of the application. We should use the thresholds to update spark configs to see DRA in play.This will help us auto-scale and utilize less resources to the max possible duration of a trigger interval.

The Query execution time upon utilization of this approach tightly hugs the scale-out threshold throughout the peak traffic hours and consumes as less executors as possible to process incoming events inside a trigger interval time, during the non-peak hours it will be get closer or below the scale-back threshold and consumes the least amount of executors required to run the app and process the incoming records.

A POC of this implementation has already been running successfully and yielded up to 66% of cost savings running structured streaming apps

Q5. Who cares? If you are successful, what difference will it make?

> Everyone who uses Spark structured streaming and has a pattern of varied incoming traffic throughout the day. This will help in savings on the costs incurred when there is less traffic and resource cycles wasted when the trigger interval is set high.

Q6. What are the risks?

- Might not be useful for streaming apps which will have multiple jobs run.
- Need to be well tested for all streaming use cases

Q7. How long will it take?

> There is already an Implementation done. Upon review and recommendation, we can decide on better approval/timelines and better testing.

Q8. What are the mid-term and final “exams” to check for success?

> The DRA for structured streaming should be available for future versions of spark.

Appendix A. Proposed API Changes. Optional section defining APIs changes, if any. Backward and forward compatibility must be taken into account.

Addition of new spark DRA configurations in addition to current spark DRA configurations.

```
Java
# enable dynamic resource allocation for structured streaming applications
```

```
"spark.dynamicAllocation.streaming.enabled": true
```

The scale out policy/threshold:

We use the same scale-out policy of a batch job but we use that to build considering the trigger Interval. If we want the streaming app to run at most 90% of the trigger interval duration (for 60s interval), we set scale out at 54s

Java

```
     # max time spark waits in each micro-batch before scaling out for
executors ( scale-out trigger)
     "spark.dynamicAllocation.schedulerBacklogTimeout": 54s
     # further delay after schedulerBacklogTimeout when the app scales out
executors exponentially
     "spark.dynamicAllocation.sustainedSchedulerBacklogTimeout": 30s
```

The scale back policy/threshold:

We utilize some of the scale-back policy configurations of the batch job and add our own. Again, for streaming workloads we consider setting the scale-back threshold at the time of the trigger interval below which we feel the cluster resources are underutilized. Eg: For a trigger interval of 60s, if we decide the cluster should be running at least half of the time we should set our scale back threshold (i.e Trigger Interval time - scale back time) to 30s

Java

```
```

```
     # time the executors wait in a micro-batch before they are scaled back by
spark app ( scale-back trigger)
     "spark.dynamicAllocation.executorIdleTimeout": 30s
     # wait time for executors which has cached data - should be >=
executorIdleTimeout
     "spark.dynamicAllocation.cachedExecutorIdleTimeout": 30s

     # percentage of idle executors to decommission on each evaluation - should
be between 0 to 1
     "spark.dynamicAllocation.executorDeallocationRatio": 0.1
     # time delay in between each executor de-allocation cycle
     "spark.dynamicAllocation.executorDeallocationTimeout": 300s
```

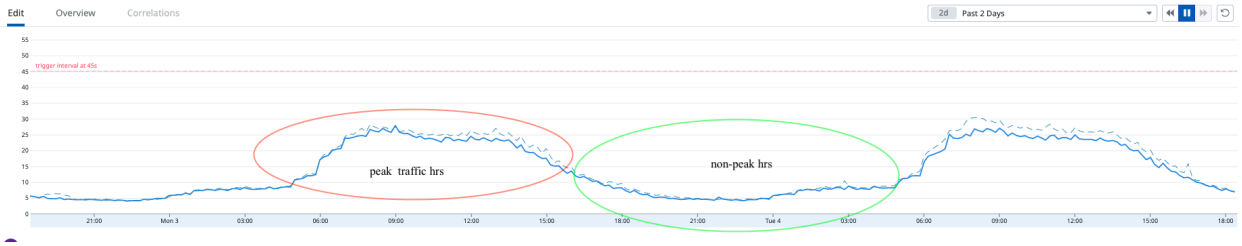
Appendix B. Optional Design Sketch: How are the goals going to be accomplished? Give sufficient technical detail to allow a contributor to judge whether it's likely to be feasible. Note that this is not a full design document.

The main need of a streaming application is to deliver/process/aggregate events in seconds/minutes latency and provide near real-time results. Streaming systems typically tend to offer low latency solutions with decent throughput by scaling out the application.

In general **latency**(lower is better) is directly proportional to **throughput**(higher is better) and inversely proportional to **cost**(lower is better). So based on the use-case we optimize, In few cases for throughput and cost we sacrifice a bit of latency. To achieve that we use a trigger Interval of secs/mins to emit records in micro-batches. We provide resources to successfully process all the incoming events in that trigger interval time throughout the day.

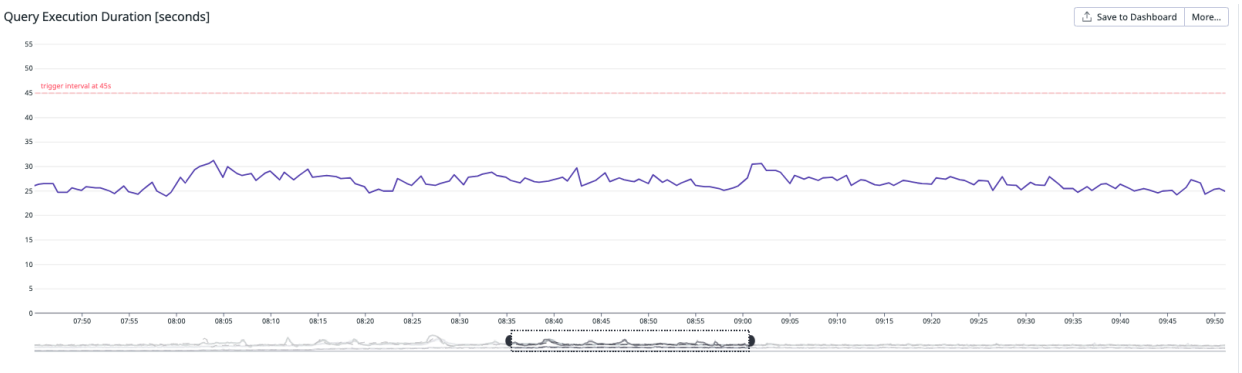
Since the Trigger interval is such an important knob for latency and throughput, we will build dynamic allocation configurations based on the heuristics of the Trigger interval.

Since in a static setup we over-provision, we typically see query(app) processing times of a micro-batch less than trigger interval. But we do observe that the query processing times are in line with typical traffic patterns of a day.



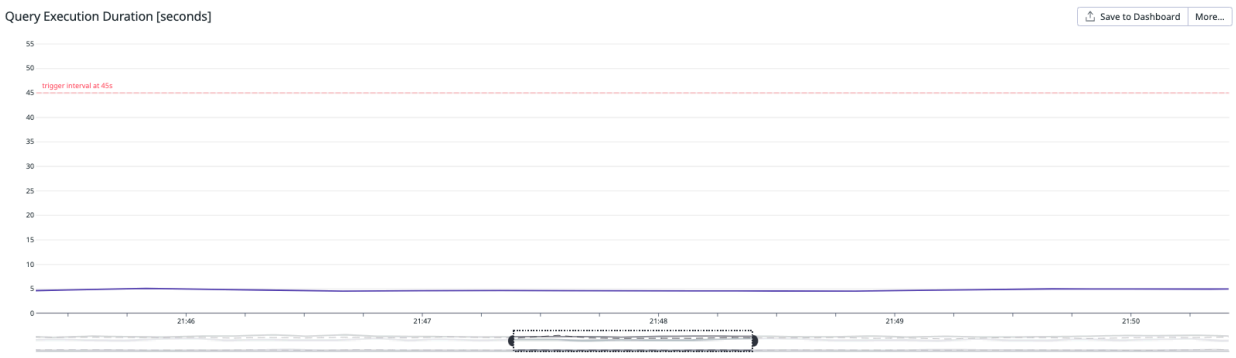
I.e the apps take longer to process a micro-batch of records in a fixed interval at morning/afternoon and less time to process records at night because we have less incoming data in that fixed interval of micro-batch.

The other reason for choosing query execution is cluster utilization of the job during that trigger interval. Let's slice the above graph to a time in peak hrs



You can observe that the query processing time (i.e the app run-time) to process the events in a micro-batch is just ~23s, so the resources allocated to it are idle for the rest of ~22s before the next micro-batch (configured by trigger interval) starts.

That is further less at non-peak hrs of the day



So Trigger interval is the right heuristic which would help us address auto-scaling based on traffic, but also better resource utilization of the allocated resources.

Generally The whole idea of scaling out with more resources is that if a job cannot process the available data it will add resources linearly(or exponentially) to handle the incoming events (i.e typical day traffic). That means in terms of a trigger interval, if the job query execution time is close to the trigger interval or not able to process all the data in the trigger interval time we need to scale out.

In the same way when a job has too many resources to process the available data it processes the data very quickly and the resources are underutilized (i.e typical night traffic). That means in terms of a trigger interval, if the job is done rather too soon then the executors are just wasting resource time until the next trigger interval.

Unset

```
How Structured streaming DRA algorithm works:

When we focus on resource utilization by trigger interval, we want the allocated resources should be used for most of the duration of a trigger interval.
So we set the scale-out and scale-back threshold limits at certain times inside a trigger interval.
To do that we should set scale-out threshold when query execution is running at an elevated/ close to
```

trigger interval time of the micro batch. The scale-back threshold should be set at a point where the query executes and sits idle for most of the trigger interval duration, thus forcing it to not hold soo many resources when it is processing data too fast.

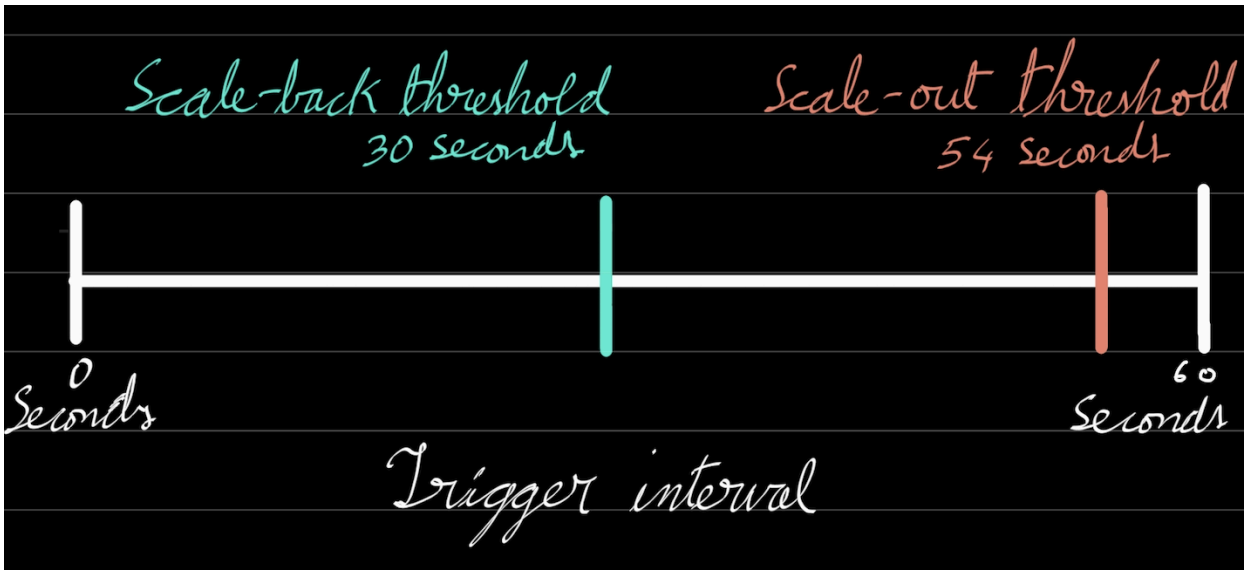
So ideally the thresholds should look like

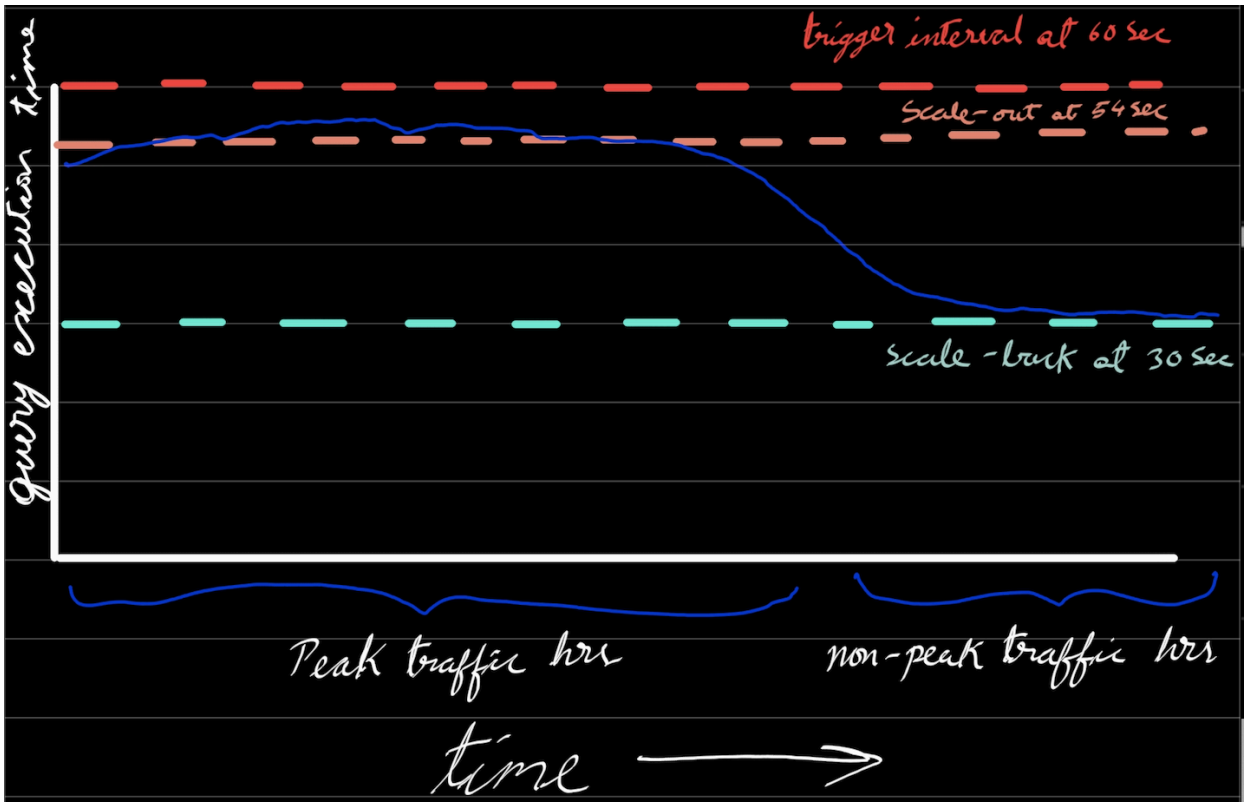
0 <= scale back threshold < scale up threshold <= trigger Interval

For eg: if we have a trigger interval of 60s

- We want the streaming application to spend at most 90% of its time (i.e 54s) utilizing the resources it has to run the micro-batch. If the query execution time is over 54s then it is very likely that the current resources it has are not enough to process the incoming data in that micro-batch so we should trigger a scale-out which will request to add additional resources (which is configurable) .

- We want the streaming application to spend at least 50% of its time (i.e 30s) utilizing the resources it has to run the micro-batch. If the query execution time is below 30s then we should trigger a scale-back which will release some of the app resources (which should be configurable).





References:

This implementation is loosely based on the discussions and concerns mentioned in https://issues.apache.org/jira/browse/SPARK-12133