Detail Design for Flow/User Level Aggregation

Introduction

We need flow/user level aggregation to present flow/user related states to end users.

Flow level aggregation involve three levels aggregations:

- The first level is Flow_run level which represents one execution of a flow and shows exactly aggregated data for a run of flow.
- The 2nd level is Flow_version level which represents summary info of a version of flow.
- The 3rd level is Flow level which represents summary info of a specific flow.

User level aggregation represents summary info of a specific user, it should include summary info of accumulated and statistic means (by two levels: application and flow), like: number of Flows, applications, resource consumption, resource means per app or flow, etc.

Related Tables

Flow_run level which represents one execution of a flow and shows exactly aggregated data for a run of flow.

Flow_Run_States Table

PKs: User, Cluster, Flow, Flow_Run ID	**Number of applications**	**Container Counters**	**Resource Metrics**	**Framework Metrics**	**Time (start, last modified)**
… + Flow_daily_analysis_0001_v1_1431113219200	applications: 3	allocated: 0 preempted: 0 failed: 0 reuse: 0	mem-allocated: 0 cpu-allocated: 0 mem-consumption: 0 cpu-consumption: 0 Resource-consumption: 0	*HDFS_BYTES_READ:* 14952678 *HDFS_BYTES_WRITTEN:* 14952678	start: 1431113229567 last modified:143111322956 7

Flow_version level which represents summary info of a version of flow.

Flow_Version_Summary Table

PKs: User + Cluster + Flow + Flow_Version	**Number of Flow_runs/ applications**	**Container counters (accumulated/mean, etc.)**	**Resource Metrics (accumulated/mean, etc.)**	**Framework Metrics (accumulated/mean, etc.)**	**Time (start, last modified)**
… + Flow_daily_analysis_0001_v1	Flow_runs: 2 Finished_Flow_runs: 1 Running_Flow_runs: 1 TotalApplications: 6 ApplicationsPerRun(avg): 3	allocated-acc: 0 preempted-acc:0 failed-acc: 0 reuse-acc: 0 allocated-avg: 0 preempted-avg:0 failed-avg: 0 reuse-avg: 0	mem-allocated-acc: 0 cpu-allocated-acc: 0 mem-consumption-acc: 0 cpu-consumption-acc: 0 Resource-consumption-acc: 0 mem-allocated-avg: 0 cpu-allocated-avg: 0 mem-consumption-avg: 0 cpu-consumption-avg: 0 Resource-consumption-avg: 0	*HDFS_BYTES_READ-mean:* 14952678 *HDFS_BYTES_WRITTEN-acc:* 14952678	start: 1431113229567 last modified:1431113229567

Flow level which represents summary info of a specific flow.

Flow_Summary Table

User + Cluster + Flow_ID	**Number of Flow_versions/Flow_runs/applications**	**Container counters (accumulated/statistical mean)**	**Resource Metrics (accumulated/statistical mean)**	**Framework Metrics (accumulated/mean, etc.)**	**Time (start, last_modification, avg_execution)**
… + Flow_daily_analysis_0001	Flow_versions: 2 Flow_runs: 4 Finished_Flow_runs: 3 Running_Flow_runs: 1 TotalApplications: 12 ApplicationsPerRun(avg): 3	allocated-acc: 0 preempted-acc:0 failed-acc: 0 reuse-acc: 0 allocated-mean (per flow?): 0 preempted-mean:0 failed-mean: 0 reuse-mean: 0	mem-allocated-acc: 0 cpu-allocated-acc: 0 mem-consumption-acc: 0 cpu-consumption-acc: 0 Resource-consumption-acc: 0 mem-allocated-mean: 0 cpu-allocated-mean: 0 mem-consumption-mean: 0 cpu-consumption-mean: 0 Resource-consumption-mean: 0	*HDFS_BYTES_READ-mean:* 14952678 *HDFS_BYTES_WRITTEN-acc:* 14952678	start: last_modification: avg_execution:

User level aggregation represents summary info of a specific user.

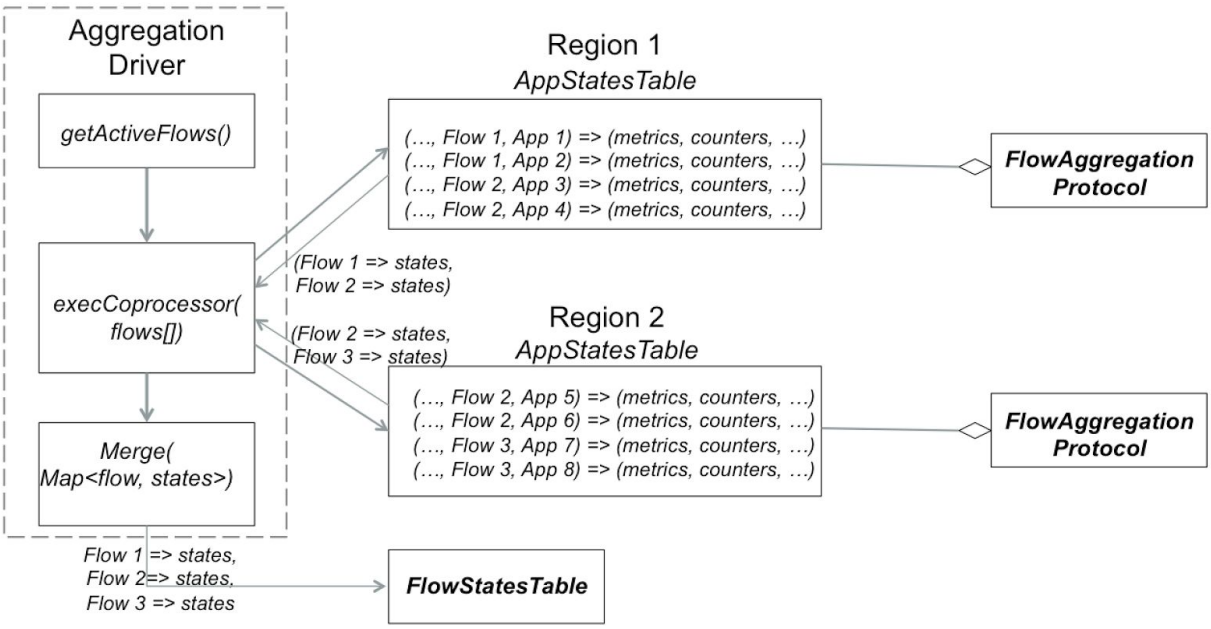
User_Summary Table

USER_ID	Number of Flows/Applications	Container counters	Resource Metrics	Cluster resource percentage (avg, peak)	Time: start, last_modification
Dr_Who_007	Flow Num: 10 Finished_Applications: 20 Running_Applications: 10 Failed_Applicaitons: 2	allocated: 0 preempted: 0 failed: 0 reuse: 0	mem-allocated: 0 cpu-allocated: 0 mem-consumption: 0 cpu-consumption: 0 Resource-consumption: 0	avg: 0.05 peak: 0.2	

Control Flow in Details

According to discussions in another document , for flow and user level aggregation, we prefer to leverage feature provided by backend storage, like: coprocessor, to aggregate info against HBase columns or column families directly across regions or even tables. With endpoints of HBase coprocessor, Flow/UserAggregationDriver (running inside of Reader daemon, will discuss this below quit soon) can trigger the endpoints periodically in a fine grained time interval (**1 min or less**) as scan rows for specific flow or user in ApplicationState table should be lower cost than queue level because current row key design (both flow_id and user_id are in key which means less regions to scan for aggregation).

This aggregation flow is as following:



We can see there are three major steps for AggregationDriver to aggregate states on flows:

1. AggregationDriver try to get active flow list to do aggregation. The definition of “active” flow is: this flow include at least one flowrun which is running or just get finished (not older than a given time window). It may need to get info of running applications from ResourceManager(s).
2. After getting flow list, AggregationDriver will trigger endpoint protocol (FlowAggregationProtocol) to aggregate states on flow at each regions. The protocol should return a map from flow to aggregated states which is half-done result but not the final one because some application states within the same flow could cross multiple regions.
3. The last step is to aggregate results that returned from FlowAggregationProtocol and write back to HBase’s *Flow_Summary* Table.

About where to run AggregationDriver, it seems several options here:

Timeline service collectors

It shouldn’t run on AppLevelTimelineCollector because the lifecycle of AppLevelTimelineCollector is bind with an application, but flow/user level aggregation could last much longer. RMTimelineCollector is not suitable too as we don’t want to handle all flow/user aggregation in a single point which is hard to scale.

New services inside of YARN daemons (RM or NM)

Either RM or NMs shouldn't handle flow/user aggregation alone. The RM-only way will make this solution hard to scale as resource pressure on single point, but NMs-only way may cause conflicts between NMs on specific flow/user aggregation (especially in case of NM failed over or restart).

Timeline Reader daemon - First step, put the AggregationDriver in Reader daemon so that we don't assume functionality in ResourceManager (suggest by Vinod)

Other Details in Performing aggregations

We assume there exists an application level aggregation table when we perform flow or user level aggregation. According to discuss above, flow and user level aggregations are implemented as an HBase endpoint coprocessor. On each time the endpoint coprocessor, or the "mapper", runs, it scans the local region on application level aggregation table, read out all new rows after its last run (we may do this through HBase filters or HBase/Phoenix secondary index).

Based on the selected data, the coprocessor then builds three Hash maps for flow level aggregations:

1. Build a hashmap indexed by user, cluster, flow, and flow_run. Then merge all timeline entities mapped to the same slot.
2. Build a hashmap indexed by user, cluster, flow, and flow_version. Then merge all timeline entities in the same slot.
3. Build a hashmap indexed by user, cluster, flow, from the hashmap built in step 1. Keep merging.

The other separated flow may build a hashmap for user level aggregation:

1. Build a hashmap indexed by cluster, user, Keep merging.

We can pass the above hashmaps to the client of the endpoint coprocessor in separated regions. Then, we can go further merge the hashmaps for different regions, and write them into HBase/Phoenix tables.

Open questions

1. Assuming we're focusing on single data for timeline metrics for now. (Reading historical data is supported in Phoenix, though.)
2. About time-based aggregations. Right now with this design, we only have one "stage", or time interval, for time-based aggregations. Shall we have daily/weekly/monthly table for cache aggregation results?
3. (**from llu**) We have some options to proceed on integrating app-level aggregation and user/flow-level aggregation:

a. If we implement the app-level aggregation table in Phoenix:
    i. For user/flow-level aggregation, we may run GROUP BY statements in Phoenix to directly aggregate on user/flow. Problem: ***Phoenix cannot handle dynamic columns gracefully, therefore we have to clearly state every metric we'd like to aggregate***. We also need to store the availability for those metrics.
    ii. We may implement user/flow-level aggregations in Hbase coprocessors, and iterate over the app-level aggregation table to generate aggregated data. Problem: ***we need to read data from Phoenix table via HBase APIs and Phoenix PDataTypes***. This is a hack and the code may not be maintainable in future.
b. If we implement app-level aggregation table in HBase only, ***we lose the ability to support SQL on app-level aggregation data***.
    i. We can use HBase coprocessors to read out data from app-level aggregation table, and then insert them into Phoenix flow/user-level aggregation tables. We need to design the coprocessors so that they can write Phoenix data for flow/user-level aggregations.
    ii. We can completely use HBase as the aggregation storage. In this way, ***we completely lose SQL query support for now***.

4. (**from vinodkv**): We have options to deploy AggregationDriver. First start with a single entity (like RM or Reader)?