Self-Service Dimensional Data Analytics

Scalable Patterns for Data-Driven Enterprises

A Technical Paper prepared for SCTE•ISBE by

Francesco Dorigo
Senior Manager
Comcast
1400 Wewatta Street, Denver, CO 80202
+1 720 512 3674
francesco_dorigo@comcast.com

Bao Nguyen
Principal Engineer
Comcast
1400 Wewatta Street, Denver, CO 80202
+1 720 512 3687
bao_nguyen@comcast.com

Daniel Howell
Senior Engineer
Comcast
1400 Wewatta Street, Denver, CO 80202
+1 720 512 3693
daniel_howell2@comcast.com

Table of Contents

Title **Page Number**

List of Figures

Title **Page Number**

List of Tables

Introduction

The IP video analytics platform design described herein streamlines the conversion of event-based analytics telemetry into time series visualization. With this objective in mind, Comcast developed a platform with the flexibility to support a wide variety of data producers and data consumers, and with the scalability to provide an enterprise solution for real-time analytics.

Parametrized and configurable execution libraries automate repetitive data engineering tasks. In addition, our analytics system provides abstraction layers both for data ingress and data egress, which enables a seamless evolution of the ETL (Extract, Transform, Load) pipeline. This has two main advantages: First to simplify the efforts related to upgrading the pipeline by letting producers and consumers evolve at their own pace, and second, the underlying technologies can seamlessly evolve with zero impact for ingestion and aggregation layers (producers and consumers).

This document describes the major components of our IP video analytics data pipeline, with a specific focus on custom components. This design reduces the time between data ingress and insight.

Our custom components are shown in Figure 1 as the collection tier (internally called “Headwaters”) and the aggregation tier (internally called “Vortex”).

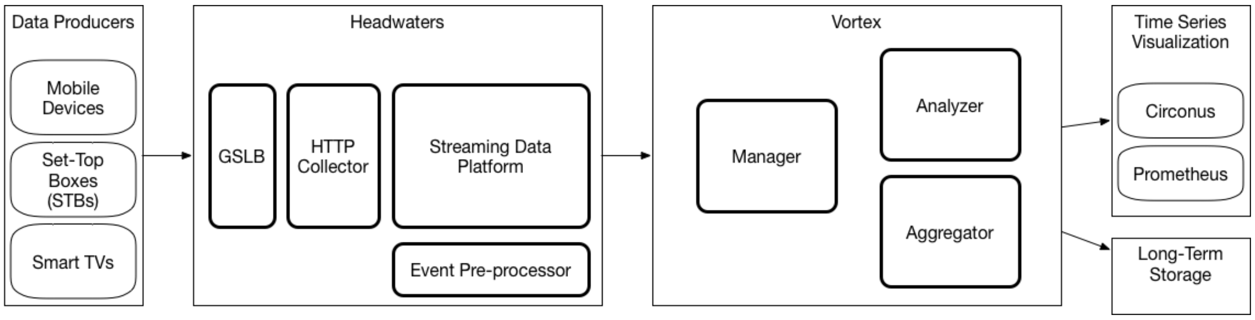

Figure 1 – Data Pipeline

Collection Tier: Headwaters

Our data collection tier was originally designed to ingest analytic events generated while streaming video content from an IP video player device. The expected growth and evolution of Xfinity TV applications call for a fast and reliable collection tier that can guarantee an actionable level of operational monitoring for system health and customers’ experience. These aspects require the collection tier to be highly configurable, reliable and scalable when adding new data sources.

The IP video analytics pipeline supports high volume, high velocity, semi structured data acquisition through a collection tier based on the HTTP protocol. The collection tier provides a REST-compliant HTTP endpoint, ensuring data extensibility, which allows most systems to natively use the REST (Representational State Transfer) architecture. For systems unable to to provide data via the REST endpoint, it is possible to deploy a thin, client-side forwarder to bridge the gap.

Under the hood, Headwaters is a streaming data platform (SDP) comprised of a REST API to receive data, and is clustered using Apache Kafka, which handles the queueing for incoming data streams. Additionally, Headwaters enforces serialization and schema governance, using Apache Avro, and provides a Confluent schema registry for event serialization/deserialization.

1. Collection Tier Components

Table 1 – Collection Tier System Description

HTTP-Collector	Exposes a REST API to clients not supporting direct Kafka APIs for publisher connections with the Kafka cluster. Performs data validation and routes data to the appropriate Kafka topics.
Streaming Data Platform	Kafka-based cluster with regional and national clusters. Allows for replication and mirroring across geographic regions/zones.
Schema Registry	Avro-based Confluent schema registry used to regulate published data content to the data exchange cluster.
EventEvent Pre-processor	Spark streaming application converting ingested data into meaningful structured data events facilitating downstream consumption.

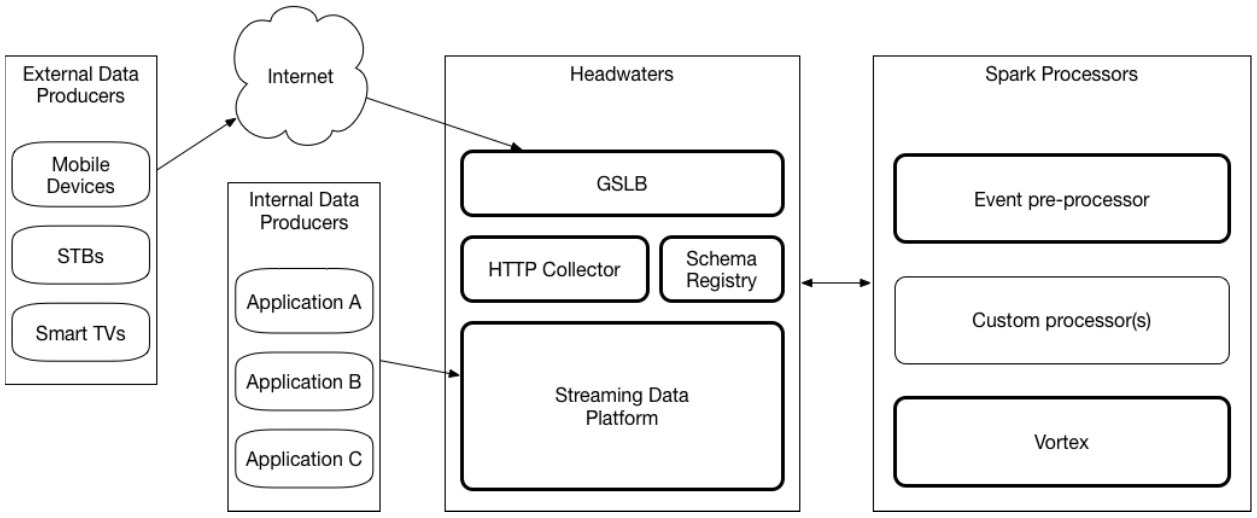

Figure 2 – Collection Tier System Diagram

At any given time, Xfinity TV applications are running on millions of devices and a wide variety of platforms (STB, iOS, Android, Roku, in-browser applications, etc.). Each of these devices are sending massive amounts of event data to the HTTP-Collector. These events represent a blend of the current internal status of the video player and the quality of the user experience as measured through predefined parameters such as startup time, bit-rate, and video format (SD/HD).

For instance, video player applications are required to send synchronous events (referred to as *heartbeats*) to periodically report current player activity and internal state. Heartbeat events are used for statistical analysis and trend forecasting and don't require immediate action from downstream consumers. On the other hand, asynchronous events are used to communicate unexpected status changes, such as warning and error conditions or user input channel changes, video playback pausing and fast forwarding, etc. The error events are dealt with immediately.

Beside the set of pre-defined events a client can send, Headwaters provides the flexibility to add new event types or include custom fields within an existing event. These customizations are performed without code changes to either the collection tier or the data consumer tier. For this reason, the architecture presented in this paper is utilized to process events generated by systems other than IP video player devices. Adding a new data source will not require any changes to the collection tier, which significantly lowers the barrier to entry for new data producers.

2. HTTP-Collector: The First Layer of Data Acquisition

The HTTP-Collector is the first layer in the data acquisition process and is bundled as a light-weight webserver. This service acts an interface for ingesting player events into Headwaters/Kafka topics where raw data is collected and transformed. The interface enables multi-tenant REST interfaces with configurable endpoints, which allows configured clients to send event data into Comcast's analytics pipeline via HTTP for processing. The HTTP-Collector is built for horizontal scale, meaning that extremely high-volume data ingestion is supported at real-time latency. Received clients' requests are structurally validated and routed into Comcast's Kafka streaming data platform.

2.1. From semi-structured to structured data

The HTTP-Collector supports the variety of player clients by allowing both generic JSON-based payloads and specific Avro-encoded payloads. In the first case, the JSON data is wrapped in a defined Avro record before being published to Kafka.

A specific consumer, the event pre-processor (detailed in section 4), is used to perform deeper validation and transforms semi-structured events into structured Avro records. In other words, the pre-processor transforms the data received as JSON into fully qualified Avro records, according to the corresponding schema, which are routed back to the appropriate Kafka topics for downstream consumption.

2.2. Multi-tenancy

Multi-tenancy is achieved by exposing different endpoints corresponding to each data producing system. In other words, when logical separation between data sets is desired, separate URLs are used to support proper data routing for data posts. For example, Comcast's syndication partners have dedicated endpoints to post event data. Each of the endpoints is backed by dedicated topics in the streaming data platform (SDP). Other systems could take advantage of a similar approach and rely on post processing systems to correlate their events with other systems' events.

3. Schema Registry and Schema Evolution

Comcast's streaming data platform (SDP) handles continuous data flow from multiple systems; each system utilizes a dedicated topic for its data stream. The Headwaters data collection tier requires an Avro schema, per topic, to enforce governance and to ensure that the content of each topic is simpler to share (content discovery). We leverage Apache Avro to serialize data and manage schemas using a Confluent schema registry to store and govern schema evolution. Avro provides rich data structures that offer a fast and compact binary data format, which allows each datum to be written with minimal overhead. The result is a more efficient data encoding and faster data processing.

Schema evolution and governance are crucial in all IP video analytics pipelines and provides the automatic transformation of an Avro schema. This transformation is only applied during deserialization. If the reader's schema is different from the writer's schema, the value is automatically modified during deserialization to conform to the reader schema using default values. Different teams and organizations within Comcast manage their own schemas for data they produce. These schemas are added, modified, or removed frequently to meet the teams' requirements, which are reviewed by a governance body before being merged. The Confluent schema registry forces schemas to be registered and associated with the appropriate topic before data can be published into Kafka.

The Confluent schema registry contains Avro schemas which are associated on a per-topic level. Each schema is used by a consumer application when de-serializing Avro event topics, because the Avro schema itself is not supplied on the Kafka event data record, only the schema ID. Our SDP Kafka topics contain Avro-serialized data only, which was previously transformed using an associated SDP Avro schema. The schema registry is responsible for serving up the associated Avro schema to provide the ability to properly de-serialize each Avro record.

4. Event-preprocessor: Data Validation and Enrichment

The purpose of the event-preprocessor is to provide correct and consistent data to downstream processing systems. The event-preprocessor is a Kafka consumer application, ensuring that the data received from the HTTP-Collector conforms to the fields defined in the Avro Schema. The event-preprocessor serializes each of the JSON events into an Avro event and decorates each with derived or sourced data from external data repositories. If the data is not conforming to the expected required fields in the Avro schema, the resulting record is tagged with warnings or errors so that downstream consumers can independently decide how they should be used.

Aggregation Tier: Vortex

5. System Overview

Data collected through the Headwaters data exchange platform is made available for any client able to consume data from a Kafka topic. Vortex is a collection of consumers, which aims to simplify the task of creating data aggregations across team domains. As a result, it saves the teams' time and shortens feature delivery, while also reducing team/system overhead. These aspects allow teams to focus on what matters most and obviates the need to create *boilerplate* data pipelines.

Data ingested by the Vortex Aggregator is targeted for structured event data, which could be specified as Parquet, JSON or Avro in this case. The data at this layer has already been cleansed by the event preprocessor (described in section 4), which is responsible for cleaning and transforming JSON data into the appropriate Avro using the proper Avro schema from the enterprise schema registry.

6. Components Diagram

Table 2 – Vortex Components

Aggregator	Responsible for processing a stream of data into multi-dimensional aggregates based on a generic configuration, the core of which hinges on an Apache Spark SQL statement. The Vortex Aggregator provides the data to the persistence layers, where the data can be served up through a REST interface for external system requests.
UI Manager	Allows for system end-users to check which jobs are running for their respective group and the associated configuration for each job. When a user wants to create a new job based on a new generic configuration, the user can use the UI and build an aggregation query, because the UI has fetched the schema registry for element selection. Configuration error validation is also performed at this layer. Lastly, the Vortex Manager enables cardinality validation. Cardinality validation ensures the level of cardinalities being asked to execute for the Vortex Aggregator are within pre-defined tolerances, which are performed by requesting cardinality counts from the Vortex Analyzer.
Analyzer	Responsible for sampling various streaming data topics to determine qualifying dimensional cardinalities. This validation acts as an execution rule for the Vortex Aggregator, since a cardinality set too large isn't considered useful for the end user. Additionally, an outcome of this cardinality processing allows the system to obtain the top-level cardinalities. For example, after Vortex Analyzer has run for a short time interval, the system understands the top X (i.e. top 3 or top 10) dimensional cardinalities, which can then be used directly by the Vortex Aggregator for processing via the generic configuration.

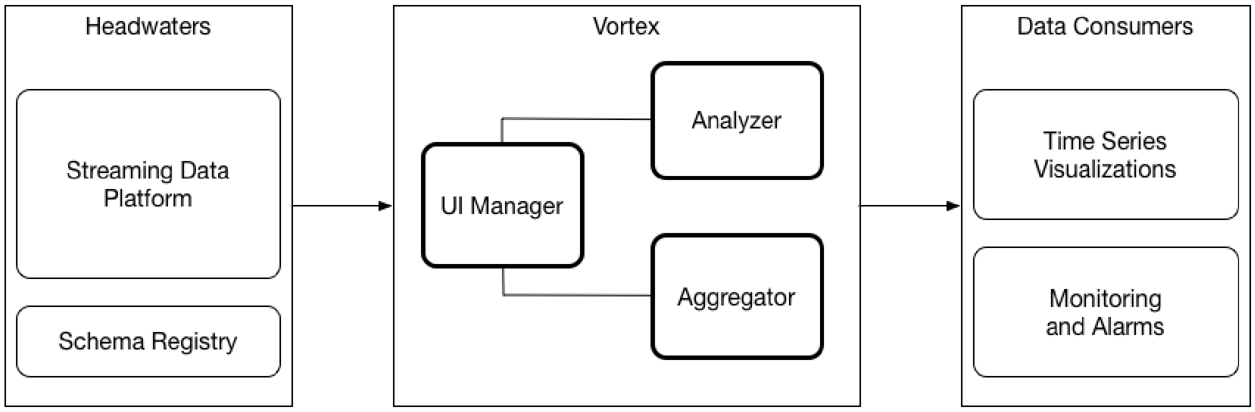

Figure 3 – Vortex Components Diagram

7. Vortex Aggregator

Vortex Aggregator is a collection of Spark applications consuming topic events from Headwaters to produce aggregations targeted for time series databases (TSDB). The Spark executables are highly parametrized so that the users can supply their specific business rules via configuration settings, rather than changing the code itself. An additional component, the UI manager (discussed later), further simplifies this task by providing a push button UI to build a generic query for a desired aggregation.

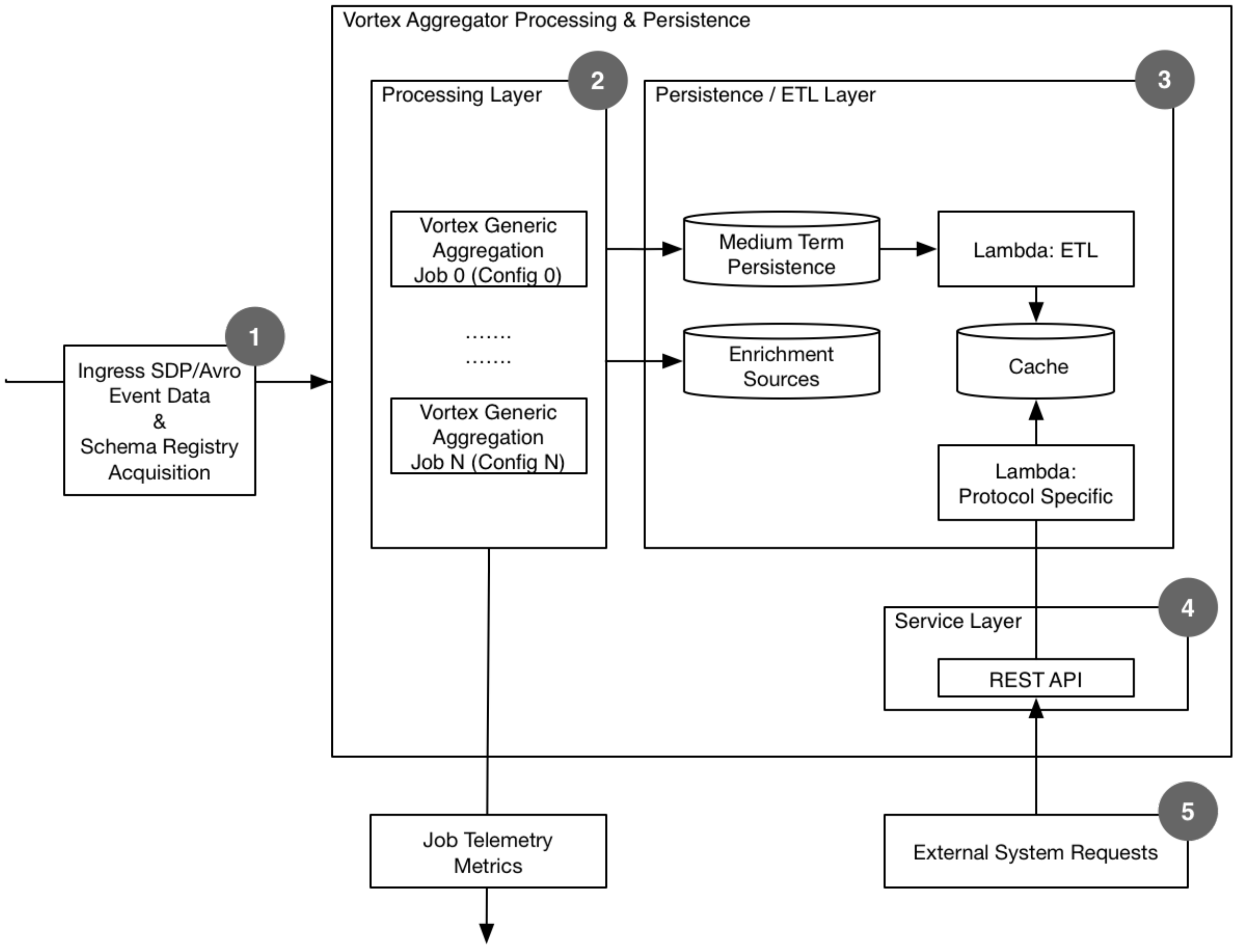
Vortex Aggregator Processing & Persistence
Processing Layer
2
Persistence / ETL Layer
3
Vortex Generic Aggregation Job 0 (Config 0)
....
....
Vortex Generic Aggregation Job N (Config N)
Medium Term Persistence
Lambda: ETL
Enrichment Sources
Cache
Lambda: Protocol Specific
1
Ingress SDP/Avro Event Data & Schema Registry Acquisition
Service Layer
4
REST API
Job Telemetry Metrics
External System Requests
5

Figure 4 – Vortex Aggregator System

7.1. Ingress SDP/Avro Event Data

As described in the previous sections, the Avro-based events are streamed from any of Comcast's SDP topics, where the events are then used to produce aggregates results. Avro events are de-serialized using the schema obtained from the Schema Registry.

7.2. Vortex Processing Layer (EMR/YARN)

The processing layer is where various Vortex Aggregation jobs execute. Vortex Aggregator is an Apache Spark application and uses a generic configuration for execution. The generic configuration relies on Spark SQL syntax to provide aggregations across multiple dimensions and cardinalities.

7.2.1. Enable Generic Multi-dimensions

Using the Vortex Manager query builder, a new generic configuration can be supplied to the Vortex Aggregator. The Vortex Manager triggers the creation of a new executable for the Vortex Aggregator. Aggregates are published within several minutes of deploying a new configuration file. This generic approach reduces the engineering effort necessary to go from "requirements" to "insights."

7.2.2. Windowing and Watermarking

Windowing and watermarking apply directly to the Apache Spark aggregation terminology. In the Vortex Aggregator, a “window” is defined as the duration of time for an aggregation using the event creation timestamp (excluding time/server adjustments). Using these timestamp values allows the proper events to be included into the appropriate window duration. Windowing is also customizable in the system.

“Watermarking” refers to the ability to handle late arriving data. The late arriving data is customized based on a user customized value to determine how late/old the data should be aggregated until those events no longer apply to the corresponding time window. This feature is useful if an application falls behind (in terms of processing), or when events are received out of order or late, to ensure aggregations are calculated properly.

7.2.3. Publish Aggregate Results

An obvious point, but worth noting, is that the goal of the system is to publish well-formed multi-dimensional aggregates into a medium-term persistence. Associating metadata with the aggregates also provides the ability to debug/trace the data from a specific configuration down to the end system, which is often a Time Series Database (TSDB). At each data touch point, timings are recorded, which provides internal latency metrics for throughput and processing speed.

7.3. Persistence/ETL Layer

The persistence layer allows for storage, transformations and data availability. These features are described below in more detail:

7.3.1. Medium Term Persistence

As a consideration of the design, aggregations were deemed valuable to publish into a medium-term persistence. These aggregate objects are kept to ensure system-to-system and configuration tracking for debugging, quality control/assurance and performance analysis. All the data at this level is a combination of the aggregation and the metadata used to generate the aggregate, including the job’s YARN application ID, configuration ID and specific topic-based data. All the metadata enables backward tracing from any point in the pipeline, including from the external systems to the Vortex Manager/user generic configuration request.

7.3.2. Enrichment Sources

An enrichment source can be considered as data which provides some degree of value for the aggregation from an additional dataset. The enrichment data at this level is considered small and can be applied using a join to help avoid verbose data shuffling across executors/partitions. For example, the joined dataset could relate to Geo Location information, which is joined on IPv4/6 addresses contained within each event. The joined data does not identify customer details but allows for decoration of market level information in real-time.

7.3.3. Lambda ETL

When a new object is created in the medium-term persistence, a trigger is fired to invoke a serverless Lambda function, which transforms the aggregate by stripping off most of the metadata. The aggregate transformation (counter, gauge & histogram data) is then written into a short-term persistence/cache. At this point, the aggregate data is ready for consumption by external systems, such as a time series database (TSDB).

7.3.4. Cache

The cache is used to hold the results for each Vortex aggregate application. When a new aggregate is created, only the most recent aggregate for the configuration ID is made available to downstream systems. This ensures that only the most recent version of the record is kept within the cache and made available to downstream systems.

7.3.5. Lambda Protocol Specific

When an external system requests aggregates from the service layer, a specific Lambda function will trigger. Each of the Lambdas are mapped directly to one endpoint (there are multiples) allowing for the same data in the cache to be transformed into a specific protocol. This flexibility allows for the same cached data to be served when called by each of the varying TSDB's where a specific format is required. An example of this conversion is when a system needs metrics in JSON vs. text-based formats supporting OpenTSDB standards.

7.4. Service Layer

The service layer provides a REST API which makes the distinct set of multi-dimensional aggregates available across one-to-many systems via HTTP GET request. This layer ensures aggregate data is extensible by design and provides the ability to transform the request into the proper response application/content-type.

7.5. External System Requests

External system requests made through the REST interface and enables:

- Time series database (TSDB), such as Circonus or Prometheus, where temporal aggregate data visualizations can be quickly created and augmented – supplying new actionable insights for a teams' workflow (alert, support, help to identify root cause, etc.)
- Any other system with the capability to make HTTP GET requests. These types of systems could be developer controlled or Quality Assurance systems for validating engineering-based changes and maintaining simulated canned test scenario performance.

8. Vortex Manager

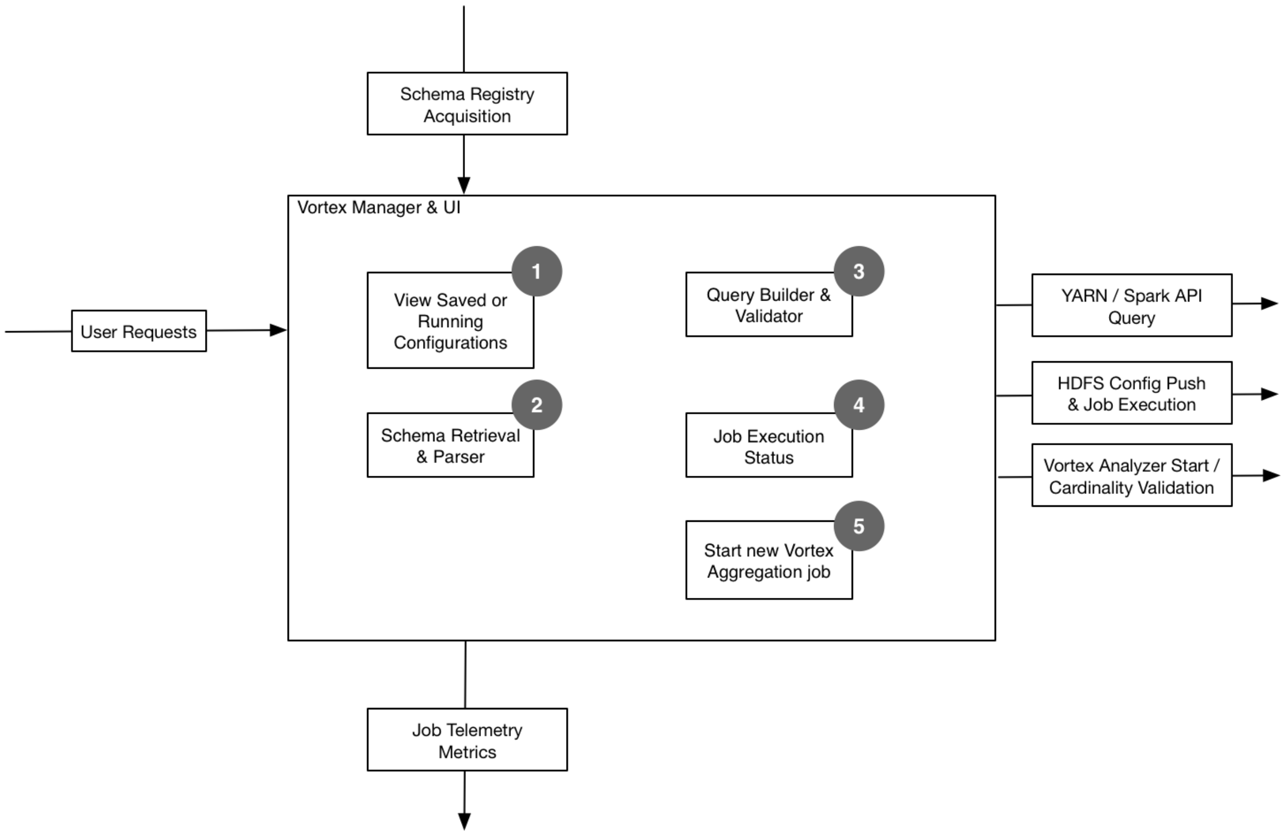
Schema Registry Acquisition
Vortex Manager & UI
1
View Saved or Running Configurations
2
Schema Retrieval & Parser
3
Query Builder & Validator
4
Job Execution Status
5
Start new Vortex Aggregation job
User Requests
YARN / Spark API Query
HDFS Config Push & Job Execution
Vortex Analyzer Start / Cardinality Validation
Job Telemetry Metrics

Figure 5 – Vortex Manager Components

8.1. View saved or running configurations

A user can view saved or running generic configurations. This helps to ensure that duplicate Vortex Aggregator applications/jobs are not executing. This is a combined view of the job status with the configurations each Vortex Aggregator application being processed.

8.2. Schema retriever & parser

Behind the scenes, the manager may already have the most recent version of the schema for a topic; however, when a user needs to browse a new topic, the manager requests the schema and parses the schema into a traversable "tree". This "tree" is then made available to the end user to point/click on one-to-many elements, which saves the user from having to know the full schema object names (which can be quite complex due to nesting.)

8.3. Query builder & validator

The query builder and validator systems allow the user to build the desired aggregation using the point/click system (additionally, other fields can be set here, such as the window and watermark durations). Users can also apply various Spark SQL syntax expressions, via filters, conditionals & SQL-based functions. Once a user's selection has been completed, the Spark SQL statement is presented for review prior to submission. During this step, validation occurs and provides areas where the generic configuration may be invalid, requiring user correction.

Once there are no further errors in the generic configuration, the user submits the request, which probes the Vortex Analyzer for information on the validity of the desired dimension and/or cardinalities. The Vortex Analyzer determines if the cardinality for the requested dimensions meet the pre-determined thresholds. This check is in place to ensure that the configuration will be useful to the end customer and ensures the Vortex Aggregator can successfully process the desired request.

8.4. Job execution status

An additional feature built into the Vortex Manager is to obtain information about the Vortex Spark applications using the YARN and Spark APIs. This information can be provided to the user as an ad-hoc request or triggered as part of the workflow to determine if an identical job is still running, in addition to identifying the job running state after submission. Additionally, the generic configurations are coupled to each current/past job.

8.5. Start new Vortex aggregation job

When the above steps meet the criterion, the configuration is stored locally for the Vortex Manager and becomes securely copied into HDFS. Once the generic configuration is uploaded, the spark-submit script (with arguments) is provided for Vortex Aggregator execution. From here, YARN/Spark manages itself by acquiring the appropriate resources and copying the files/code to each of the executors for execution.

In the meantime, the Vortex Manager supplies the user with information about the status of the job using the typical YARN states. Once in "RUNNING" state, various information is recorded about the job and associated back to the user's request.

9. Vortex Analyzer

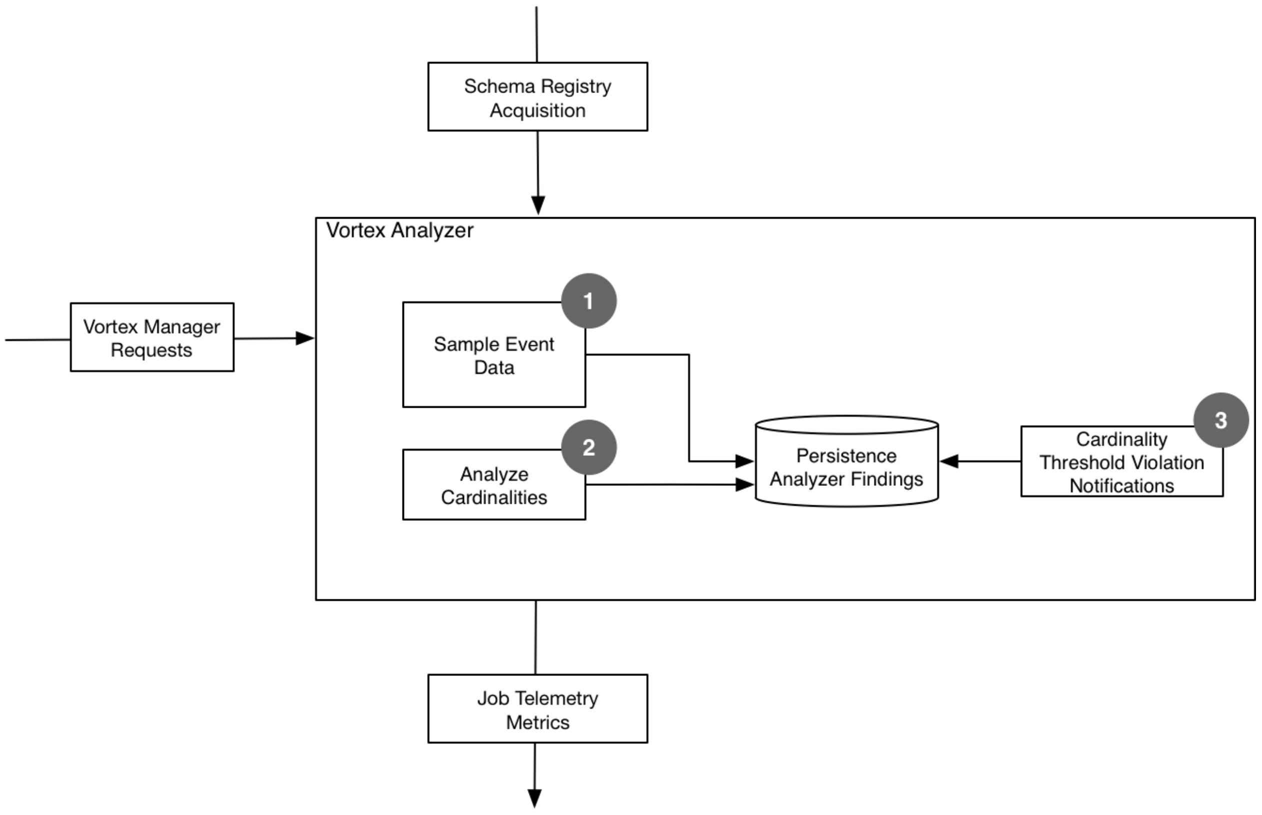
Schema Registry Acquisition
Vortex Analyzer
Vortex Manager Requests
1
Sample Event Data
2
Analyze Cardinalities
Persistence Analyzer Findings
3
Cardinality Threshold Violation Notifications
Job Telemetry Metrics

Figure 6 – Vortex Analyzer Diagram

9.1. Sample event data

Vortex Analyzer samples data for various topics to record the dimension and cardinality state (number of occurrences, i.e., depth of the dimensions). When the Vortex Analyzer runs in “normal” mode, it executes in the terms of hours per day, which is enough time to record cardinality state. “Quick” mode offers a fast inspection of the event topic data to determine the same dimension and cardinality state, which is a tradeoff between time to execution vs. confidence levels.

When an unsupervised Vortex Analyzer topic is requested, a user can apply two execution states – “quick” or “normal” -- both resulting in a new Vortex Analyzer job but with differing degrees of results. In quick mode, the Vortex Analyzer will run for just a matter of minutes, and, based on those findings, a validation decision will be made to determine if the Vortex Aggregator can execute for the requested dimensions and cardinalities. Quick mode allows the user to get aggregates flowing as quickly as possible for a new “Headwaters” topic. In normal mode, the same analysis is performed, but provides a much higher level of confidence for the witnessed data. Once the execution time has passed (usually measured in hours), the validation results will be returned to the Vortex Manager with a callback and will allow the user to then submit the job. The execution of the job can be performed autonomously, so the user doesn’t need to wait for the execution of validation results.

When either mode is selected by a user, the option exists to enable the Vortex Analyzer to run continuously, because other validations may be requested in the future, which makes the topic supervised by the Vortex Analyzer. All quick mode jobs will be scheduled into normal mode jobs when a user requests for continuous validation on the specific topic. This is a feature of the scheduler built into the Vortex Analyzer.

9.2. Analyze & Validate cardinalities

Dimensional cardinalities are examined to ensure the breadth of the data fits within the pre-defined tolerances and doesn't violate the expansion rules. The rules in place provide the ability to limit jobs from running thousands of cardinalities, which wouldn't be useful from a TSDB visualization perspective. However, across large data sets, there may be a need to examine only a handful of dimensional cardinalities. The Vortex Analyzer provides Vortex Manager with the top "X" dimensional cardinalities. This allows a filter to be applied for the Vortex Aggregator and enables a level of control that wouldn't have been possible by using a simple click from the Vortex Manager.

For a user to enable this feature, a selection in the Vortex Manager is applied for the top "X" data points available, where "X" is scalable up to a bounded limit. This way, the user can understand the top values for large datasets and encourage exploratory analysis using other tools within the data analysis ecosystem.

9.3. Cardinality threshold violation notification

One additional feature of the Vortex Analyzer is to analyze cardinality threshold violations for executing Vortex Aggregator jobs. This background processing handles post analysis so as to deeply analyze acquired data, which identifies once "thought to be good - valid" configurations to "bad - invalid." Such an occurance could be caused by an upstream application release.

To account for this possibility, the topic data is continuously supervised/analyzed. When a threshold violation is triggered, a notification is sent to the manager (and to the internal telemetry metric system, which is distinct from external TSDB's). The Vortex Manager will indicate the problem with a level of confidence for the possibility of failure/impact. Additionally, the telemetry metric system will deliver an alert as part of SRE/OPS support. If the job fails, the application service manager for YARN/Spark (a standalone background component) will not attempt to restart the job, because it now violates the pre-defined rules of an acceptable configuration and requires a user to re-issue and validate the requested configuration. Typically, when this edge case occurs, a user can submit a new generic configuration using the maximum top X, which typically removes the violation. This feature provides a technique to protect the Vortex Aggregator and the user's end system from large degrees of data drift and cardinality explosions.

10. Example of Vortex Application

As data consumers add more and more dashboards to monitor their systems, they discover the inherent value of visualizing the data being collected. Correlations between measurements that belong to the same system enable deeper exploratory analysis. Real-time execution of aggregations shortens the mean time to detect (MTTD) and react to operational issues.

In the example below, several platforms were experiencing an unusually high spike in error rate per video playback start. The operation and engineering team was alerted immediately, and focused on the platform with the highest error rate, diagnosed the problem, and resolved it in a timely fashion. Figure 7 shows the session starts for iOS, desktop, and Android platforms in purple. The daily trend exhibits the expected peak near the prime-time hours of the day. The error rate overlaid on the graphs below in green shows a sudden and sustained spike for all platforms.

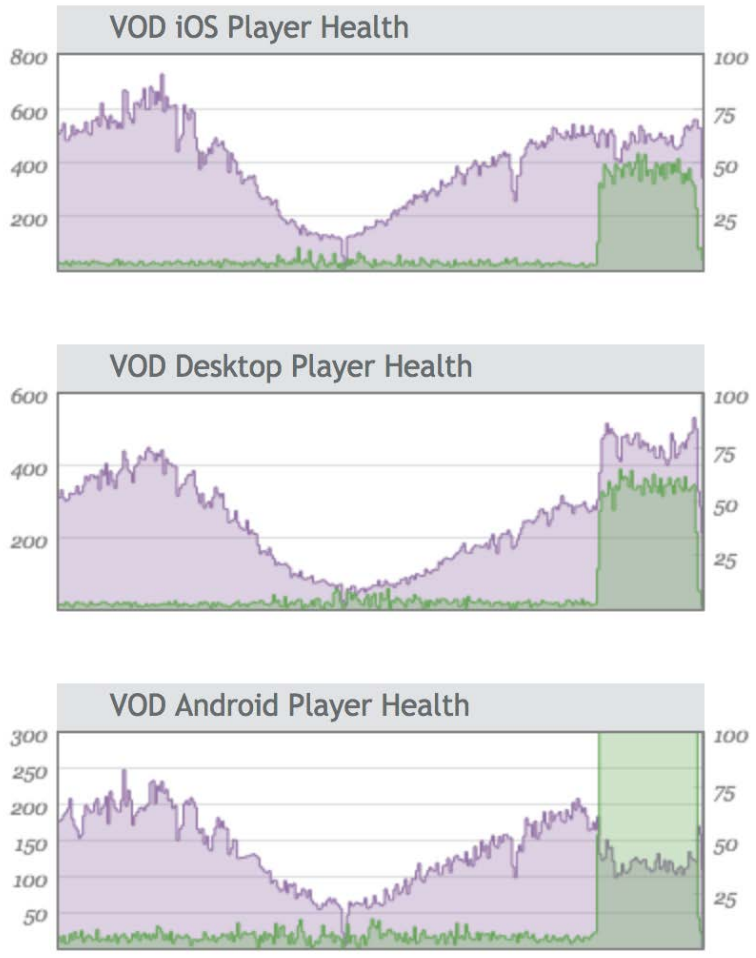

Figure 7 - Anomaly detection in example

Conclusion

We invest heavily in growing and expanding the analytics capabilities of all components of the IP video delivery platform. This document summarizes the current architecture for a state-of-the-art and end-to-end data pipeline that can scale horizontally to adapt to the growing needs of the enterprise. Moreover, automating and simplifying the data engineering tasks required to aggregate and visualize event-based telemetry provides a low risk migration option for systems that have outgrown their own ad-hoc solutions.

A universally available REST API for data ingestion, a scalable data stream platform, a push button aggregation system, and powerful time series visualization tools are key elements for the successful evolution of a data exchange platform.

As an enterprise data exchange solution, Headwaters and Vortex provide unprecedented data sharing opportunities for all organizations within Comcast. The systems and practices described in this document reduce the effort necessary to collect, prepare, and share the data between internal groups. Having a common solution for most applications allows us to focus all data engineering resources on improving the performance and feature offerings of the data exchange, rather than providing a dedicated ad-hoc solution for each system.

Abbreviations

ETL	Extract, transform, load
SDP	Streaming data platform
STB	Set-top box
TSDB	Time series database