Paper SAS6431-2016

Modernizing Data Management with Event Streams

Evan Guarnaccia, Fiona McNeill, Steve Sparano, SAS Institute Inc.

ABSTRACT

Specialized access requirements to tap into event streams vary depending on the source of the events. Open-source approaches from Spark, Kafka, Storm, and others can connect event streams to big data lakes, like Hadoop and other common data management repositories. But a different approach is needed to ensure latency and throughput are not adversely affected when processing streaming data, that is, they need to scale. This talk distinguishes the advantages of adapters and connectors and shows how SAS® Event Stream Processing can leverage both Hadoop and YARN technologies to scale while still meeting the needs of streaming data analysis and large, distributed data repositories.

INTRODUCTION

Organizations are tapping into event streams data as a new source of detailed, granular data. Event streams provide new insights in real time, helping organizations improve current situational awareness, respond to current situations with extremely low latency, and can improve predictive estimates for proactive intervention.

When an organization begins to use streaming data, or when it decides to enhance its real-time capabilities and offerings, many things need to be taken into account to ensure that scaling to the high throughput (hundreds of thousands of events per second and more) can be achieved.

An easy place to start would be the three Vs of big data: Volume, Variety, and Velocity. As the Internet of Things continues to grow, a natural increase in the volume and variety of data follows. Sensors have become smaller and cheaper than ever and it makes sense that businesses would want to take advantage of this detailed data to monitor activity more closely and on shorter time scales, thereby leading to an increase in the velocity of high volume streaming data. Customers are interacting with businesses in ways they never have before, such as through apps, social media sites, online forums, and more. This has led to unstructured text becoming an important and valuable source of data along with more traditional types of data. Variety isn't just associated with unstructured content in event streams but also relates to the different formats of data emanating from sensors, applications, and machines – transmitting event data at different intervals, formats, and levels of consistency.

SAS Event Stream Processing offers the flexibility, versatility, and speed to be able to tackle these issues and adapt as the landscape of the Internet of Things (IoT) changes. Whether a single event stream or multiple event streams are ingested for insights, scaling to this fast and big data will separate those organizations that are successful in putting event streams to use for organizational advantage from those that become swamped by the pollution from their overflowing data lakes.

WHAT INSIGHTS SCALE WITH EVENT STREAMS

With continuous data delivered in event streams, the ongoing evaluation of conditions, activities and interpretation become possible. Differences between what was and what is are compared. Outliers are apparent. Patterns are found. And trends, toward or away from normal conditions are proactively assessed.

Comparison to historic conditions and statistical measures (like average, standard deviation, and alike) are defined relative to tolerance thresholds. This comparison can be done for unique events, or collections of events. For collections of events, live streamed data is aggregated and held in memory for comparative purposes. These aggregations are referenced windows that compile and compare event summaries and their corresponding elements to new data that streams through the event stream continuous query.

The ability to retain event data, and calculate in-memory conclusions from it, can be conceptually

compared to a short, retained history of live activity – that is, events that have just occurred in a limited window of time (of course, based on live data, versus data that's been stored). This means that standard data manipulation and analysis tasks that require more than a single event value can readily be calculated in live, streaming data using aggregate windows. Given this, a host of insights and tasks can be accomplished in streaming event data – before it is even stored to disk – promoting scalable insights to even the most complex, big, and high throughput streaming data sources.

SCALING DATA QUALITY

Treating event streams as a new source of data and simply storing it – to be examined later – pollutes data lakes with irrelevant, incorrect, and unusable data. Initially, such pipeline processing might not be considered much of a problem, given the low costs of commodity storage and technologies to directly investigate data stored in Hadoop.

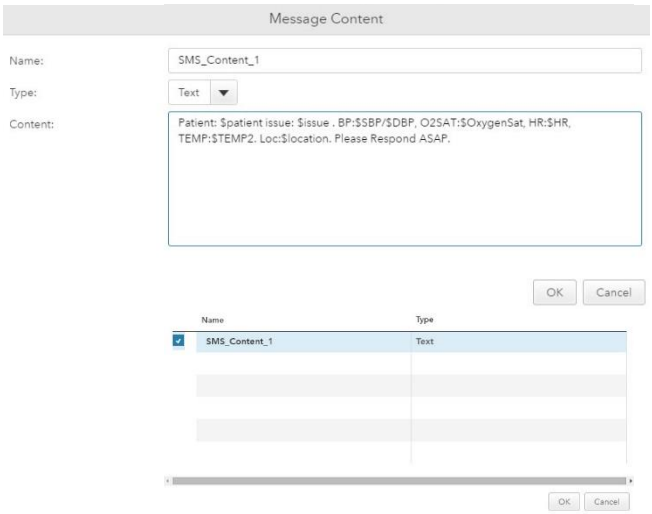
Open-source pipeline processing is a current data strategy used by organizations to ingest event streams data and store in Hadoop or other big data repositories. However, with the accelerating volumes of event data, generated in the IoT and other technologies, the volume will soon compound even the lowest, incremental costs. It's expected that IT will soon be called to task to reduce these growing incremental costs. A longer term, more strategic approach is needed, one that won't overflow and pollute data lakes, fill clouds with data waste and crippled data centers with irrelevance.

Streaming data, even from the most regulated environments – like that of sensors – will contain data errors. Gaps in events, from network interruptions and erroneous measures that have no merit are commonplace. Correcting data streams as they are ingested reduces downstream processing data manipulation needs and helps create data that's ready for use.

SAS Event Stream Processing is used to filter out unwanted or unimportant event data and even correct it while it's still in motion, before it is stored. Determinations made on live data are invoked using data quality routines available in SAS® Data Quality and Natural Language Processing (NLP) from SAS® Text Analytics. These routines cleanse data, identify and extract entities, categorize unstructured data, and help manage event records. When these routines are applied directly to streaming data, they have the following capabilities:

- Create a normalized standard from an input that is context specific
- Correct nonstandard or duplicate records as well as identify unknown data types
- Separate values through natural language parsing
- Identify and resolve entities using logic types (phone number, name, address, location, and so on)
- Extract common entities from unstructured text
- Validate data against standard measures and customized business rules
- Categorize unstructured text
- Determine case and gender
- Generate unique match codes
- Identify sentiment, and more.

Applying such data quality and text analytics routines within the data stream, before it is stored, fills in the gaps of missing data, corrects event data errors, and standardizes inputs and formats. The routines also help to determine if events are meaningful and are of further use in analytical investigation. If they are not of further use, no further processing is required. Unwanted data is filtered out, and the process saves the incremental cost of storing that data. You don't store what you'll never need, and you can substantially decrease further network transmission of event data as well as downstream processing costs.

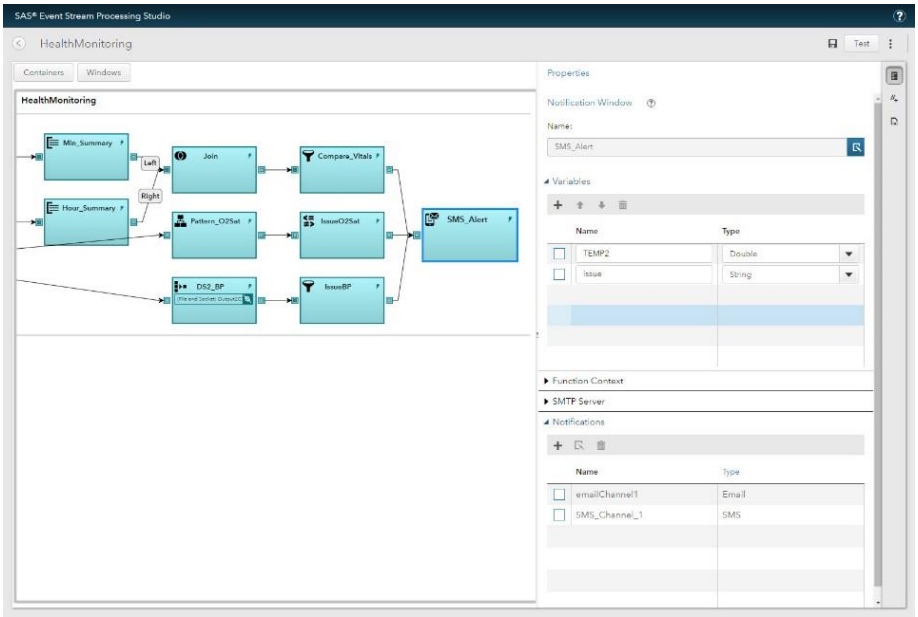


SCALING ANALYTICS

One goal of analytical event stream processing is to have current situational awareness to existing conditions – for example, to ask: Are current events outside of normal operating parameters? As such, continuous queries often focus on the changes of events, those that deviate from normal conditions. If all is normal, then no further action is required. However, if events aren't normal then real-time alerts, notifications and actions are issued to further investigate or react to such abnormal activity. Within SAS Event Stream Processing, you can detail the alert conditions, message and recipient information – directly in the Studio interface, as illustrated in Figure 1.

Figure 1 - SAS Event Stream Processing Studio

Figure 1 shows alerts included in stream processing (right), detail alert channel and recipients (upper) and real-time condition details (lower).



Knowing the appropriate conditions of when a pre-defined tolerance threshold is crossed is defined by the rules in the system. For example, in statistical process control, the Western Electric Rules[1] stipulate decisions as to whether a process should be investigated for issues in a manufacturing or other controlled setting. These rules signify when an event, or calculation based on an event are relevant. This relevance can be highlighted in dashboards, to trigger operational processes or alerts that are sent to other applications listening for these events of interest. Any combination of rules and analytical algorithms are possible with SAS Event Stream Processing, so you can devise and adjust your scenario definitions, tolerance threshold levels defined as rules directly in the studio interface.

SAS Event Stream Processing also allows historic data (called into memory) to be assessed in tandem with live event stream processing for evaluations based on summary conclusions that have been made from existing knowledge bases, like a customer segmentation score. Such lookups based on data that has been stored in offline repositories (also known as "data at rest") opens another suite of conditional assessments that can be made from streaming event data, like last action taken, pre-existing risk assignment, or likelihood of acceptance.

SCALE TO ALL PHASES OF ANALYTICAL NEED

Event stream processing is also used to get an accurate projection of the future likelihood of events, to answer questions such as "Will this machine part fail?" In doing so you can better plan, schedule, and proactively adjust actions based on estimates of future events.

One differentiating feature of SAS Event Stream Processing is the ability to use the power of SAS advanced analytics in stream. This is done using a procedural window, in which users can write their own input handlers. Currently, input handlers can be written in C++, SAS DATA step, or SAS DS2. When SAS DATA step is used, calls to Base SAS are made each time an event is processed. When SAS DATA Step 2 or C++ is used, no calls are processed, which expedites processing times.

Creating the predictive algorithm outside of streaming data, built on a rich history from existing repositories, including historic events and then scoring events using the algorithms as part of a continuous query (via the procedural window) is one method to scale analytics to streaming events. Many analytical questions require rich history in order to identify the best model for prediction.

Some data questions, however, don't require extensive history to define an algorithm. This is the case when models learn as new data arrives, and when learning does not require historical data. For these types of data questions, the equation or algorithm can be defined in the event stream. SAS has introduced K-Means clustering as the first iteration of advanced analytics learning in the data stream (again, with no reliance on previously training the model using stored data). As illustrated in Figure 2, sourced live events are first processed by a training stage, where events are examined by the algorithms, and then events are scored using that algorithm, with no calls out to any dependent processing. As stream events change, the model updates to new conditions, learning from the data. The K-Means streaming algorithm is defining clusters of events, which group events into homogenous categories. Other, such machine learning algorithms, are planned for future releases of SAS Event Stream Processing.

[1] More on Western Electric Rules here: https://en.wikipedia.org/wiki/Western_Electric_rules

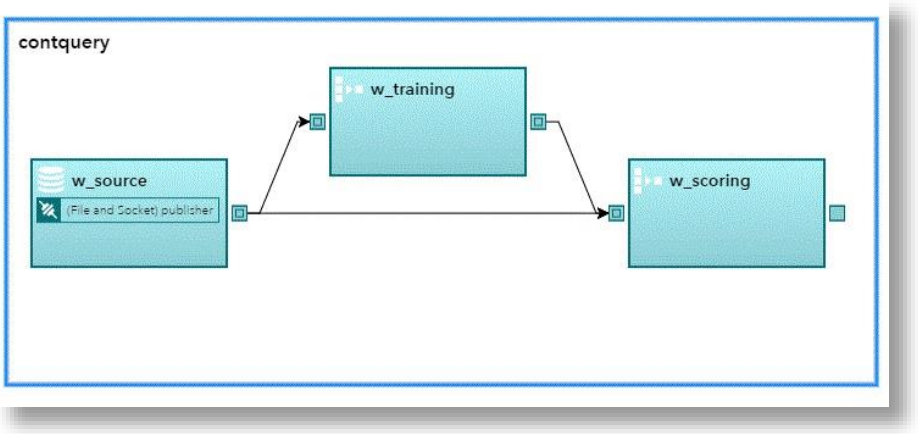


Figure 2 - Learning Algorithms that Score Streaming Events in SAS Event Stream Processing Studio

As an embeddable engine, SAS Event Stream Processing can also be pushed to extremely small compute environments – out to the edge of the IoT, like compute sticks and gateways. At the edges of the IoT, and even individual sensors, event stream data is limited by the transmitting or gateway device and as such, some analytical processing will make sense while algorithms that require a variety of inputs, won't. As you move away from the edge, more data is available for more complex decision making. And out-of-stream analysis, based on stored data, like Hadoop, will have extensive history upon which investigation and analytic development can happen. SAS Event Stream Processing can be used throughout, pushing data cleansing, rules and machine learning algorithms to the edge, learning models in-stream and including score code from algorithms developed from data repositories, scaling to all phases of analytic need.

HOW SCALING WORKS

Latency and throughput are major concerns that need to be addressed when a business decides to implement real time, streaming data solutions. Latency is the amount of time it takes for an event to be processed from start to finish. Throughput is measured by how many events can be ingested per second. Scaling to reduce latency and to ingest increasing throughput is a balance that needs to be based on the business requirements for the application at hand.

SCALING FOR LOW LATENCY

While some sources of latency are common to all deployments of SAS Event Stream Processing, there are many different ways to architect a real-time solution, and with each comes different sources of latency. Common sources of latency include the following:

- processing wide records
- complex processing steps
- making calls out to external systems
- using stateful models.

Wide Data

It is very common for streaming data sets to be wide with many fields per record, especially when multiple event streams are all ingested into the same processing flow. Since the time it takes to process an event scales linearly with the number of fields, it makes sense to eliminate fields as early as possible that will not be used for any useful purpose. This can easily be done in a compute window, where the user can

specify which fields from the previous window will be used going forward. In addition to reducing the number of fields to be processed, compute windows can also be used to change data types and key fields. Ideally, a compute window is defined directly after the source window (the latter, which ingests event stream data), so that minimal time is spent processing unneeded event fields.

Reduce Complexity

SAS Event Stream Processing is able to perform complex operations on data, but sometimes this is at the cost of latency. String parsing is an example of an operation that can increase latency. Such processing can be done using the SAS® Expression Language or using user-defined functions written in C. Using the SAS Expression Language is simpler to use than a custom function, but custom functions typically run faster.

Managing State for Lower Latency

As an in-memory technology, SAS Event Stream Processing is dependent on available memory for processing speed. Each window in a model can be assigned an index type, making it a *stateless*, or a *stateful* window. A stateless window ingests an event, processes it, and passes it on without retaining the event data in memory. To ensure low latency, models should be kept as stateless whenever possible. Sometimes, however, state needs to be retained, such as when streaming data is joined to a lookup table, when aggregate statistics are being calculated, or when pattern windows accumulate partially matched patterns.

Very often, streaming data can be enriched by joining it to historical records, customer data, or product data. At times, these lookup tables can be very large - to the point where it is undesirable or even impossible to keep the entire table in memory. When this happens, the HLEVELDB index should be used for the local index of the dimension side of a join. The HLEVELDB index type stores the lookup data on disk and provides an optimized way to retrieve such records, effectively offsetting the latency that typically comes with using data that isn't stored in memory.

Another common scenario is to retain statistical information about the streaming data for additional processing, using an aggregate window. This raises the issue of event retention. To ensure that memory consumption is bounded, a retention policy must be implemented in which the user defines the number of retained records or for how long records are retained. This is accomplished by preceding an aggregate window with a copy window. Also, the pattern window also needs to retain some state associated with events – best understood by describing how the pattern window works.

Pattern Compression

The user defines a pattern of interest, which will most likely consist of multiple events. When an event arrives to the pattern window, the pattern window holds that event while it waits for other events that comprise the pattern of interest. As the number of these partially matched patterns increases, memory usage can grow quite large. The impact of this can be offset by enabling the pattern compression feature. By compressing unfinished patterns, memory usage can be reduced by up to 40% with the cost of a slight increase in CPU usage.

SCALING FOR HIGH THROUGHPUT

For an event stream processing solution to scale to increasing volumes of data, it must be designed in such a way as to handle high throughput. There are many factors to consider because of the varied and diverse use cases.

Managing Thread Pools

Each SAS Event Stream Processing engine contains one or more projects, each of which has a defined thread pool size. This enables the project to use multiple processor cores, which allows for more efficient parallel processing. Often, streaming data will have to travel over a network to reach a SAS Event Stream Processing server. In that case, the throughput is limited by the speed of the network connection. A typical 1GB/sec interface should be able to process about 600 MB/sec of event data. To achieve a higher

throughput, projects can be spread across network interfaces. One option is to connect SAS Event Stream Processing projects in a star schema, where many projects are taking in data from the edge, aggregating it down to desired elements and performing any preprocessing - all connected to a central continuous query, which ingests the prepared data and performs the desired operations. It should be noted that retaining state in source windows can affect throughput.

Events are grouped into event blocks consisting of zero or more events when they are first ingested using a source window. Using larger event blocks helps increase throughput rates during publish and subscribe actions. Event blocks can, at times, only contain one event, such as when aggregate statistics are being joined with incoming events. If an event block contains an insert and multiple updates, they would be collapsed to a single insert containing the most recent values. In this case, the aggregate statistics would not accurately reflect the stream of incoming events.

SAS ESP and Hadoop Technologies

As the adoption of Hadoop continuous to grow, it's important for technologies like SAS Event Stream Processing to deeply integrate with Hadoop – to make the best use of its features. It might be desirable to architect a streaming solution with aspects of the event stream processing model residing on different machines. In some instances, one might want to separate data preparation from pattern matching because of memory constraints. The same holds true for parts of the model that join streaming data to large tables of dimension data. In order to reduce latency and increase throughput, SAS Event Stream Processing models need to be designed in such a way as to take full advantage of a distributed environment.

Hadoop uses a resource management platform known as YARN to allocate resources to applications. YARN uses containers, which represent a collection of physical resources, and SAS Event Stream Processing servers can be run in these containers. With SAS, one can specify the memory and number of cores to be used in each container. This means that more memory and parallel processing power can be allocated to parts of the overall event processing model that are more computationally intensive. Assuming that network connectivity is present, outside users can also connect to ports opened by the SAS Event Stream Processing XML servers in the Hadoop cluster. From the perspective of the outside user, then, the functional behavior will be the same as if SAS Event Stream Processing was being run stand-alone.

SAS EVENT STREAM PROCESSING ADAPTER/CONNECTORS

SAS Event Stream Processing provides Hadoop adapters for integration with large data sources to both read and write data (events) to these distributed data targets. But simply storing and writing efficiently to these systems is only part of the story to deliver an adaptive and scalable system. SAS Event Stream Processing provides necessary integration with the YARN resource manager on Hadoop to leverage the resource management capabilities for higher throughput by leveraging the distributed processing framework in Hadoop.

YARN, which was designed as a generic resource negotiator platform for a distributed system like SAS Event Stream Processing, uses the same underlying daemons and APIs that are within the common ecosystem of other Hadoop applications. SAS Event Stream Processing nodes run directly within the fully managed YARN environment thereby leveraging the power of the YARN resource manager, as illustrated in Figure 3. To leverage YARN's resource management power and deliver maximum throughput, SAS Event Stream Processing provides a YARN plug-in to communicate with the YARN environment and submit the necessary requests for cores and memory needed for performance.

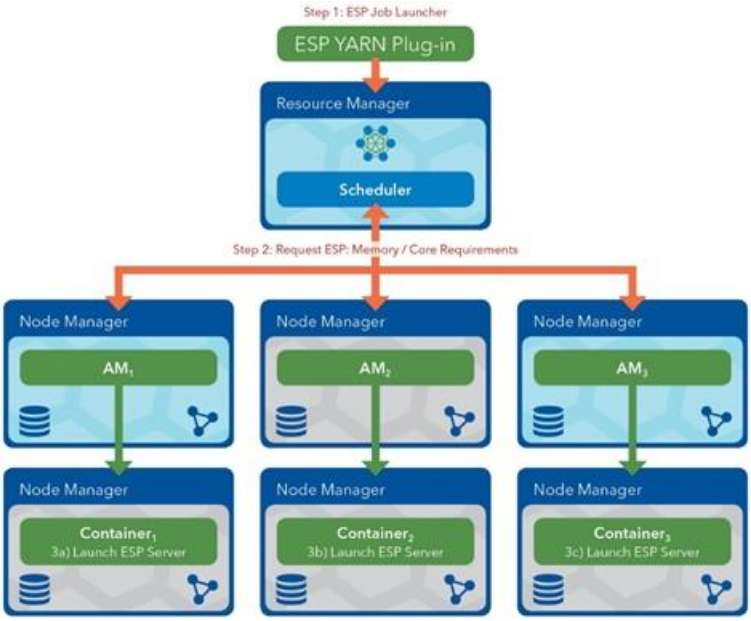


Figure 3 –SAS Event Stream Processing Integration with YARN.

As shown in Figure 3 - SAS Event Stream Processing server on Hadoop, the ESP application has been started and is running using three YARN containers and is managed by the YARN Node Manager. This allows YARN to manage the ESP servers running in the various nodes to control start up and shut down for a seamless and scalable processing environment and the requested nodes can be increased when additional processing resources are needed.

The YARN plug-in supports commands using dfesp_yarn_joblauncher - for requesting and launching YARN cores and memory, as noted in Figure 4.

```
lin64:yihuan@solace02:~/install/lin64/SASEventStreamProcessingEngine/4
.1.0/esp-qa/regressions$ $DFESP_HOME/bin/dfesp_yarn_joblauncher -e /r/
sanyo.unx.sas.com/vol/vol920/u92/yihuan/install/4.1.0 -a 61000 -t 6100
1 -u 61002
```

Figure 4 - Launch SAS Event Stream Processing on YARN

Using the SAS Event Stream Processing Application Master interface, show in Figure 5, the SAS Event Stream Processing XML factory server, “qsthdpc03”, is shown running in the YARN-managed Hadoop environment and will be using the noted http-admin, pubsub, and http-pubsub ports, as well as the requested virtual cores and memory.

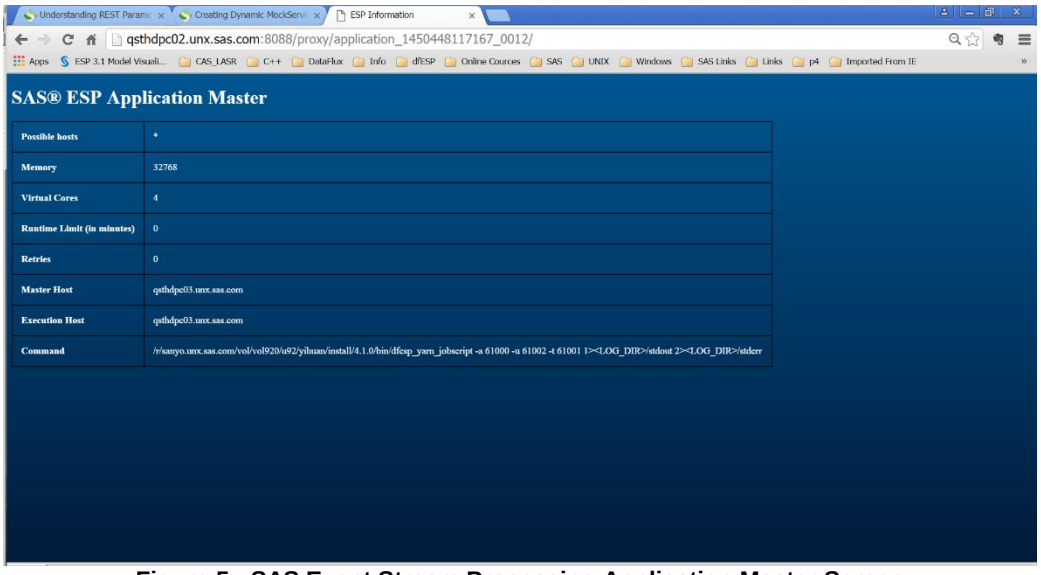


Figure 5 - SAS Event Stream Processing Application Master Screen

By using the http-admin port defined for the running ESP server, “qsthdpc03”, commands are used to load ESP project “test_pubsub_index” through dfesp_xml_client into the running ESP XML factory server, depicted in Figure 6.

```
lin64:yihuan@solace02:~/install/lin64/SASEventStreamProcessingEngine/4
.1.0/esp-qa/regressions/hadoop/yarn_joblauncher/runJoblauncher$ ${DFES
P_HOME}/bin/dfesp_xml_client -url "http://qsthdpc03.unx.sas.com:61000/
SASESP/projects/test_pubsub_index?overwrite=false&connectors=false" -p
ut "file://xml/load_project.xml"
<message>load project 'test_pubsub_index' succeeded</message>
lin64:yihuan@solace02:~/install/lin64/SASEventStreamProcessingEngine/4
.1.0/esp-qa/regressions/hadoop/yarn_joblauncher/runJoblauncher$
```

Figure 6 - SAS Event Stream Processing Project Load Example

To deliver additional processing for higher throughput, a second SAS Event Stream Processing factory server is started. This environment can be discovered and managed using the consul service to monitor its performance characteristics. As shown in Figure 7, a second server, “qsthdpc02” is running and available as an additional resource within YARN for event processing.

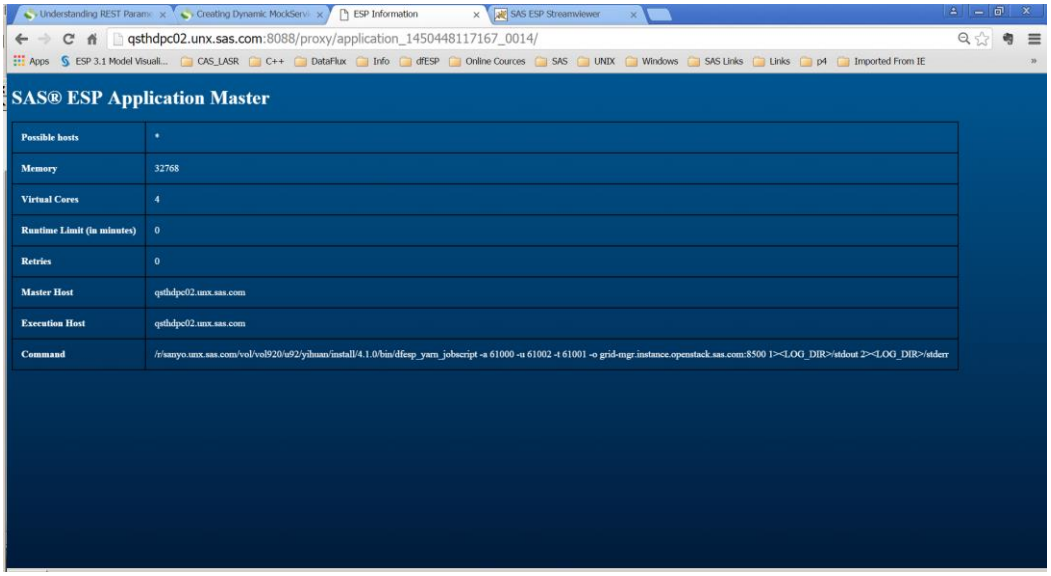


Figure 7 - SAS Event Stream Processing Application Master running an Additional Server for Higher Throughput

The consul service view provides the health check information about the publish, subscribe, and HTTP Admin interfaces for “qsthdpc02” as well as other SAS Event Stream Processing servers running and managed by the YARN resource manager on Hadoop, as shown in Figure 8.

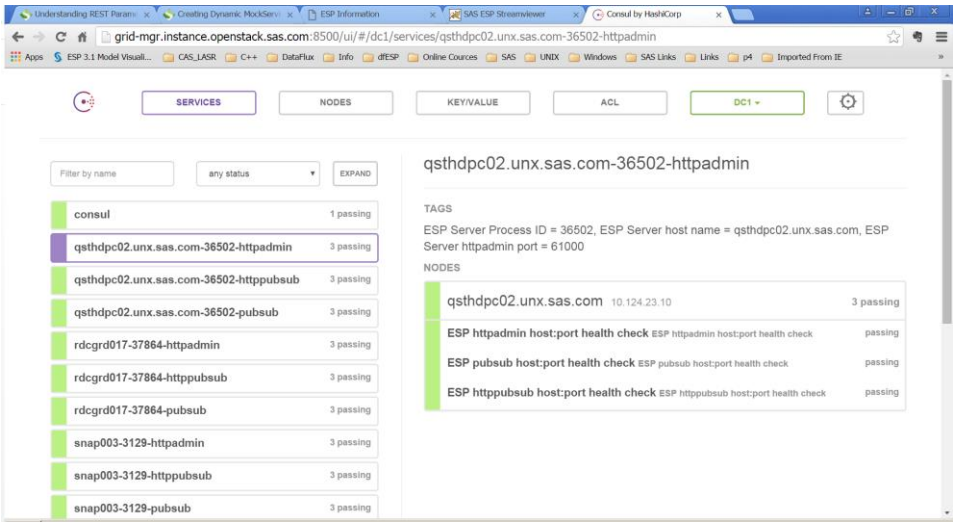


Figure 8 - SAS Event Stream Processing Status within Consul Interface

ENTERPRISE CONSIDERATIONS

With the approaches outlined above, we can see how SAS Event Stream Processing supports scaling to meet the processing demands of the enterprise for latency as well as throughput without sacrificing either. These important performance considerations are not the only factors that need to be considered for scalability. In any technology there are the other overriding factors that are needed to deliver reliability, productivity, flexibility, governance, and security.

A complete ecosystem for managing and governing the code for a successful streaming analytics environment is needed for enterprise applications. Many open-source tools provide components for what is needed to deliver aspects of streaming performance but tend to lack the capabilities needed for complete business solution. A complete business solution includes scalability, reliability, and governance – aspects that ensure a solution supports the enterprise's needs both today and tomorrow, scaling to new problems and data volumes.

RELIABILITY

Today's IT infrastructures require that event streams are processed in a reliable manner and are protected against any loss of data or any reduction in IT's service level agreements.

SAS Event Stream Processing provides a robust and fault-tolerant architecture to ensure minimal data loss and exceptionally reliable processing to maximize up time. SAS does this by delivering reliability using proven technologies - like message buses and solution orchestration frameworks – that ensure message deliverability while eliminating performance hits on the SAS Event Stream Processing engine. This translates to a solution that is reliable and which supports failover. One definition of failover is:

> "switching to a redundant or standby computer server, system, hardware component or network upon the failure or abnormal termination of the previously active application, server, system, hardware component, or network."
>
> *Reference: https://en.wikipedia.org/wiki/Failover*

Failover architectures are essential to any system that demands minimal data loss. SAS Event Stream Processing has a patented approach for a 1+N Way Failover architecture, as illustrated in Figure 9.

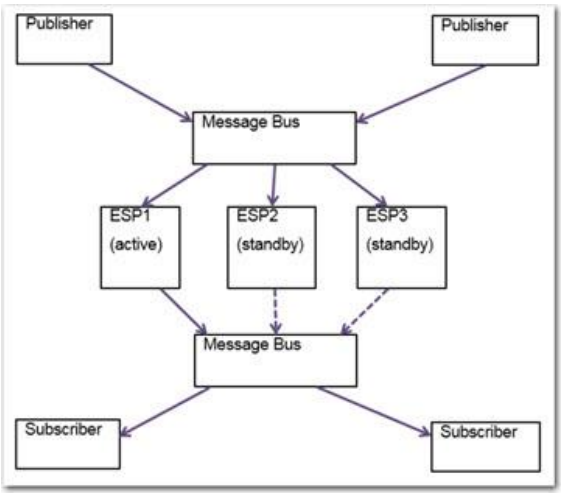


Figure 9 - SAS Event Stream Processing 1+N-Way Failover

As shown in Figure 9, the failover architecture allows the SAS Event Stream Processing engine, subscribers, and publishers to be oblivious to failover and any occurrences thereof. The product pushes the failover responsibilities and knowledge to the (prebuilt) publish and subscribe APIs and clients. The APIs and clients are, in turn, complemented by third-party message bus technology (from Solace Systems, Tervela, or RabbitMQ). This architecture has the benefit of flexibility, as it allows failover to be introduced without requiring the publishers and subscribers to be changed or recompiled.

In this approach, “N” (in 1+ N-Way Failover) refers to the ability to support more than one active failover at a time, and as such, you can be assured of getting as close to zero downtime as can be afforded. All event streams are published to both the active and stand-by SAS Event Stream Processing engines as part of standard processing. Only the active SAS Event Stream Processing engine forwards subscribed event streams to the message bus (for subscribers). If the message bus detects a dropped active connection or missed “I’m alive” signal, the message bus then appoints a stand-by to be active with the event block IDs to begin for the subscribers. This new active engine keeps a running queue for subscribed event streams. The queue is then used to start forwarding events to the subscribers.

SAS has another patent pending for *Guaranteed Delivery*. This informs a publisher callback approach when event blocks are received by one or more identified subscribers within a configured time window. The same callback function is notified if this does not occur, so the publisher determines how to handle that situation. This is done asynchronously and without persistence so as not to impact performance. As a result, failover is instantaneous and automatic with no loss, nor replay of events, no performance degradation, and a reliable environment that meets IT’s needs.

PRODUCTIVITY

Developing models to ingest, analyze, and emit events can be a complicated task when you consider the various window types, the sophisticated analysis, testing the model, and reporting results. All of this is required to ensure that the model produces the desired actions. SAS Event Stream Processing provides tools to assist with the development of models for processing event streams.

Model Design

SAS Event Stream Processing Studio is a design-time environment that supports the development of engines, projects, and continuous queries. These components form a hierarchy for the model building environment, as illustrated in Figure 10. The studio is one of the three ways to build a model in SAS Event Stream Processing.

SAS ESP has three modeling approaches that are 100% functionally equivalent that provide developers

the flexibility they need to develop, test, and implement streaming models. These approaches include:

- C++: a C++ library that can be used to build and execute ESP engines.
- XML: XML syntax to define ESP Engines or Projects via an XML editor.
- Graphical: SAS Event Stream Processing Studio is a browser-based development environment using a drag-and-drop interface to define ESP models, either engines or projects.

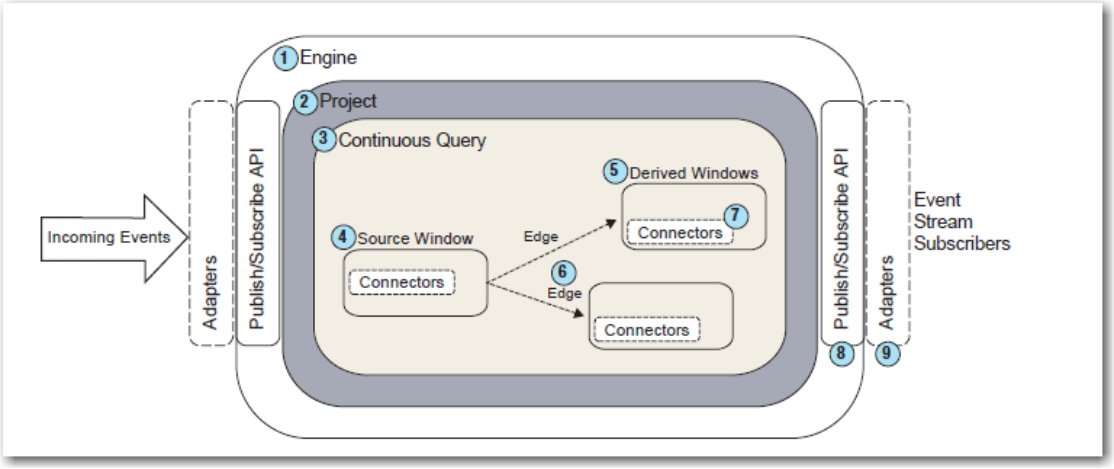


Figure 10 - SAS Event Stream Processing Model Hierarchy

Using the hierarchy depicted in Figure 10 - SAS Event Stream Processing Model Hierarchy, sophisticated event stream processing models are defined. The top level represents the engine, and within an engine, one or more projects can be created allowing for flexibility in how the events are processed. Different projects can be coordinated to allow events to be delivered from one project to another project for processing. Finally, within a project, the SAS Event Stream Processing Studio interface supports continuous queries, where events are processed using the window types available from a menu (window types are shown in Figure 11).

Typically, SAS Event Stream Processing Studio is used to quickly build streaming models that include the flow of data from source windows through to the processing windows for pattern matching, filtering, aggregations, and analytics. The drag-and-drop interface supports rapid event stream model development and doesn't require any XML or C++ coding to deploy these models. Of particular note, the Procedural window is how SAS analytical models are introduced into the event streaming data flow.

Once the design is complete, the user can test the models from within the interface.

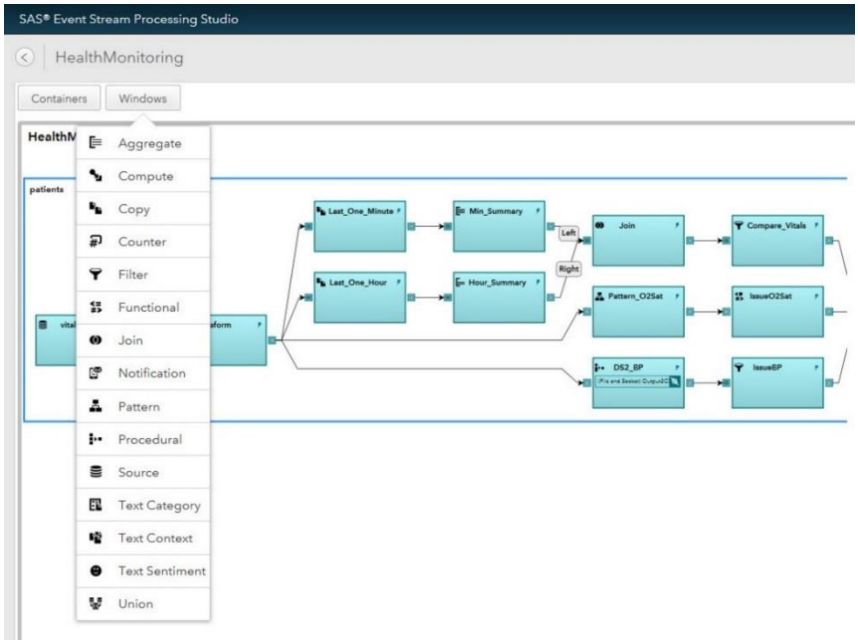


Figure 11 - SAS Event Stream Processing Window Types

Testing and Refinement

SAS Event Stream Processing Studio provides an interactive test mode, which can be used to load event streams into the continuous query, and publish results from select windows to the screen once the event streams are processed by the model. This functions as an easy to use diagnostic tool that aids the model builder during the development and testing phases.

SAS Event Stream Processing Streamviewer provides additional insights into the streaming model behavior with overlays of intuitive graphics that visually depict trends in the rapidly moving data. This is essential to understand model performance in high throughput data. As illustrated in Figure 12, the interface provides a view into the live results of each window by simply subscribing to any window of interest – so that testing can identify if the model matches expected results.

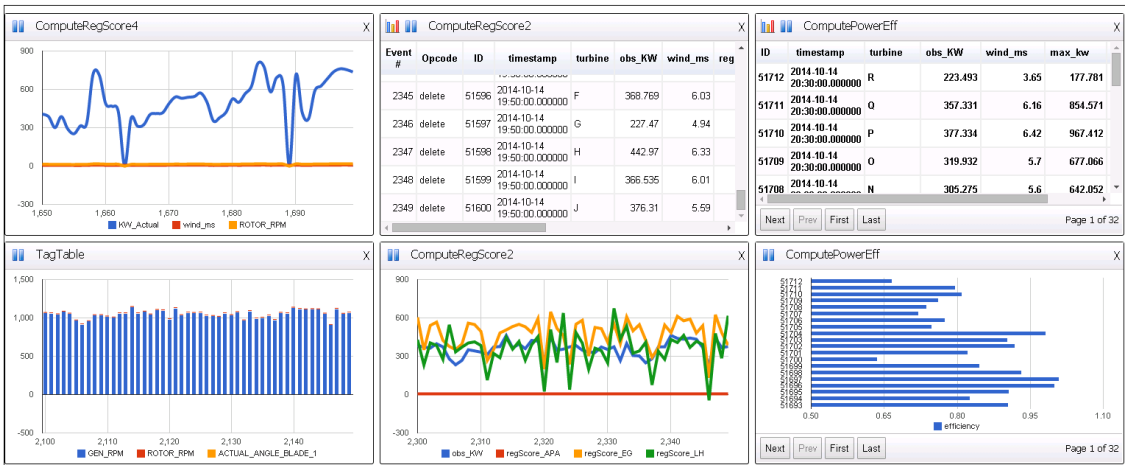


Figure 12 - SAS Event Stream Processing Streamviewer

SAS Event Stream Processing Streamviewer allows for rapid model iterations by visualizing the trends in the streaming data, and eliminates the need to build a custom visualization tool for testing.

FLEXIBILITY

Given the flexibility and power of the engine, continuous queries, and streaming window types, streaming models can be complex designs that introduce branching, left and right joins, schema copies, pattern matching, and analytics. All of these moving parts can be difficult to orchestrate, and as with any complex design, can be difficult to communicate to other teams in the organization. This can introduce delays and risk as teams struggle with describing stream processing designs to other groups in a way that ensures all parties understand the solution as well as their specialized involvement. Additionally, when skilled resources are scarce, and design logic expertise is in short supply, a visual representation of a model achieves a common, clear and effective means to communicate a complex model design to others. SAS Event Stream Processing visual Studio is often valued by teams, providing an easily consumable format for design specification, thus reducing such risks.

The Power of a Visual Interface

As shown in Figure 13, SAS Event Stream Processing Studio is a graphical event stream model design-time environment, making it easier to share designs between stakeholders, using the export and import feature. This allows models to be shared across design environments. This graphical design-time environment also allows new staff to quickly understand the streaming model definition, reducing the risk associated with only a few staff possessing the knowledge of how a particular event stream processing flow operates.

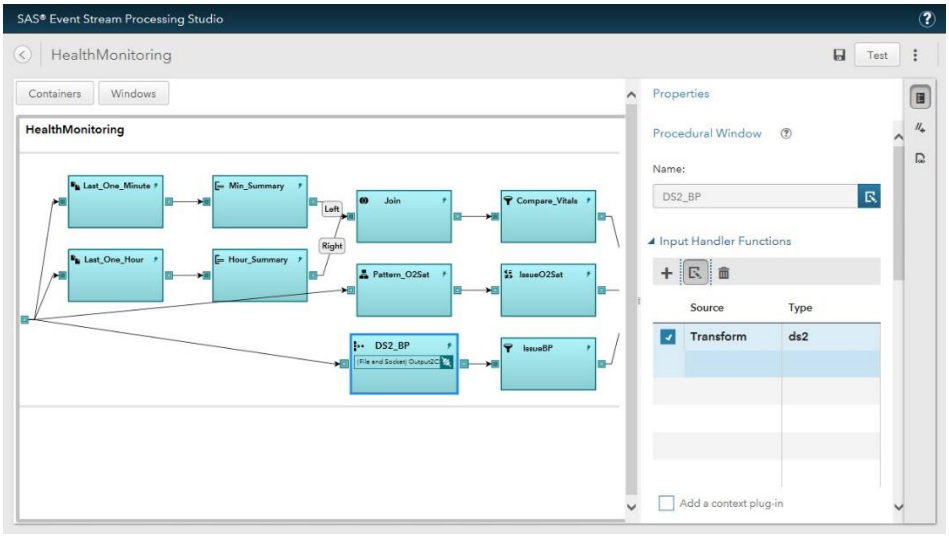


Figure 13 - SAS Event Stream Processing Studio Graphical Editor

Evolving Stream Data

Adapters

Data types and streaming data sources are constantly evolving and it is important to have the ability to quickly adapt, and this includes the ability to include new streaming data sources and types. The SAS Event Stream Processing Publish and Subscribe API allows extensibility for new adapters to be built using the same API used to develop the prebuilt adapters. These APIs include both JAVA and C APIs. A wide array of adapters are supplied out of the box (as listed in

Table 1), which can be further configured or custom adapters can be built using the same Java or C++

API. Adapters can also be networked to allow for coordination between different input and output streams of data.

API Language	Adapter
C++	Database Event Stream Processor File and Socket IBM WebSphere MQ PI Rabbit MQ SMTP Subscriber Sniffer Publisher Solace Systems Teradata Subscriber Tervela Data Fabric
Java	HDAT Reader HDFS (Hadoop Distributed File System) Java Message Service (JMS) SAS LASR Analytic Server REST Subscriber SAS Data Set Twitter Publisher

Table 1- SAS Event Stream Processing Studio Adapters (C++ and Java)

Connectors

Similarly, SAS Event Stream Processing connectors can be also created and integrated using the Java and C++ APIs. Connectors are “in process”, meaning that they are built into the model during design time. In contrast, adapters can be started or stopped at any time, even remotely. Connectors use the SAS Event Stream Processing publish/subscribe API to do one of the following:

- publish event streams into source windows. Publish operations do the following, usually continuously:
    - read event data from a specified source
    - inject that event data into a specific source window of a running event stream processor
- subscribe to window event streams. Subscribe operations write output events from a window of a running event stream processing engine to the specified target (usually continuously).

The SAS Event Stream Processing Connectors include:

- Database
- Project Publish (Inter-ESP Project)
- File and Socket
- IBM WebSphere MQ

- PI
- Rabbit MQ
- SMTP Subscribe
- Sniffer Publish
- Solace Systems
- Teradata Connector
- Tervela Data Fabric
- TIBCO Rendezvous (RV)

Each of these connectors supports various specific formats.

Taken as a whole, this collection of streaming data connectors and adapters offers a robust collection of pre-built routines to ingest streaming data and deliver outputs. The connectors also offer an extensible framework to use new event stream sources.

Given that each data management approach is different, SAS Event Stream Processing supports large data stores subscribing to fast-moving streams. The data can be landed in distributed file systems like Big Insights, MapR, Cloudera, and Hortonworks.

GOVERNANCE

Deployment of event streaming models to various targets requires version control, configuring publishing targets, and updating models dynamically to minimize interruption of service to both event publishers and subscribers.

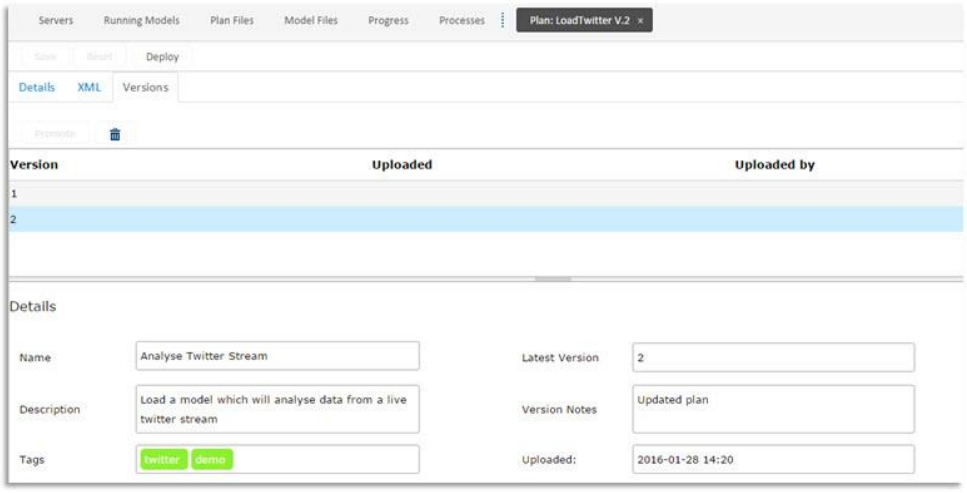


Figure 14 - SAS Event Stream Processing Studio Change Management

SAS Event Stream Processing provides support to manage versions and to publish changes to models as illustrated in Figure 14. Changes to models can be scripted using plan files that not only support changes to the deployed streaming models, but also coordinate the loading of the updated model to a running SAS Event Stream Processing engine. Also addressed are the orchestration of adapters to inject events to the updated model and validation that the model is syntactically correct.

This support is enabled by the configuration of XML plan files (see Figure 15) that manage these changes at publish time. These plan files enable reuse for streamlined operations and governance as well as ensuring flexibility for managing multiple models. This automation allows for repeatability across operational scenarios for consistency and repeatability.

Figure 15 - SAS Event Stream Processing Example Plan File

In conjunction with the support for plan files to update and publish new models, developers can use the SAS Event Stream Processing engine's dynamic service change feature to change models on the fly without taking the SAS Event Stream Processing server down - ensuring constant up-time for always-on streaming applications. Specifically, users can add additional windows, remove windows, or change windows as part of these dynamic updates. Dynamically changing models on a running XML factory server without bringing down the project or significantly impacting service processing improves business agility.

SAS Event Stream Processing manages such dynamic changes without losing existing state, where possible, and can propagate retained events from parent windows into newly added windows. If the new streaming model design changes a given window, then most likely the state is no longer meaningful and will be dropped.

The implication is that new analytic score code can be dynamically updated into deployed streaming models as the need arises, so that analytics are refreshed on an as-needed basis while governed in a controlled manner.

SECURITY

Data streams can include sensitive data, requiring that data be secured when in flight as well as during processing. This, in turn, necessitates that the publishers (delivering data to SAS Event Stream Processing), the subscribers (to the processed events), as well as event data stored in-memory during processing be secure. SAS can secure event data when in-memory from unauthorized access.

The prior release of SAS Event Stream Processing, version 3.1, provided encryption of data streams both to and from the SAS Event Stream Processing engine (both publish and subscribe) using OpenSSL when communicating between client and server. The OpenSSL option is available when using the SAS® Event Stream Processing System Encryption and Authentication Overlay provided with the product.

SAS Event Stream Processing 3.2 introduced an optional authentication between the clients and servers to ensure more secure access to the product's network interfaces such as the XML server API and the Java/C Pub/Sub APIs and adapters. This was also extended to SAS Event Stream Processing Streamviewer.

CONCLUSION

The best utilization of Hadoop and other big data lakes for streaming data is achieved when a strategic

approach is adopted – one that doesn't pollute them with dirty or irrelevant noise. Direct integration with YARN helps scale for higher throughput by using the distributed processing framework of Hadoop. Scaling to examine streaming data once it is landed in a big data repository is, however, only one consideration when scaling for big and fast data.

There is a balance to be struck between what is best done as part of event stream processing before event data is stored, and what is more appropriately done once it is landed in Hadoop. Many advanced analytical models require a rich history to appropriately model the desired behavior. Hadoop, as a popular big data environment, is ideal for in-depth SAS analysis – and often appropriate to build and define SAS DATAStep2 score code to be embedded with SAS Event Stream Processing.

SAS Event Stream Processing can ingest, cleanse, analyze, aggregate, and filter data while it's still in motion – helping channel only relevant big data to such 'data at rest' repositories for in-depth diagnostics and investigation. Event streams can be assessed as they are sourced, filtering out irrelevant noise, saving network transport loads, and focusing downstream efforts on what's relevant.

Configuring a solution to address high throughput volumes with low latency response times, that can successfully ingest data streams and provide the answers needed by the business, is dependent on both the infrastructure environment as well as the event stream processing model itself. Using the SAS Event Stream Processing integration with YARN enables a dynamic linkage of these two technologies. This further scales the power of the service management of YARN in Cloudera, Hortonworks and other Hadoop environments – while SAS reduces the data stream data, generates immediate insights and balances resources for an optimized business solution.

For enterprise adoption, additional scaling considerations that go beyond those of in-memory environments, analytics, throughput, and latency are also core to successful event stream processing deployments. With an ever changing business climate, event stream processing applications also need to be reliable, productive, flexible, governed, and secure. SAS Event Stream Processing provides the agility needed to scale – for organizations who are extending their existing SAS knowledge by tapping into the new sources of insight that event streams provide, and scaling to those already tackling the IoT frontier.

ACKNOWLEDGMENTS

The authors would like to acknowledge the guidance and assistance of SAS colleagues Jerry Baulier, Scott Kolodzieski, Fred Combaneyre, Vince Deters, Yiqing Huang, and Yin Dong for your direction and support of this paper.

The authors would also like to acknowledge that Figure 3 of this paper was jointly crafted in partnership with Hortonworks – initially defined to illustrate SAS Event Stream Processing YARN integration with the Hortonworks Data Platform.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Evan Guarnaccia
SAS Inc.
Evan.Guarnaccia@sas.com

Fiona McNeill
SAS Inc.
Fiona.McNeill@sas.com

Steve Sparano
SAS Inc.
Steve.Sparano@sas.com