Paper SAS563-2017

Managing Real-Time Data Streams to High-Performance Analytics Engines

Katherine Taylor, SAS Institute Inc.

ABSTRACT

You're in the business of performing complex analysis on large amounts of data. This data changes quickly and often, so you've invested in a powerful high-performance analytics engine with the speed to respond to a real-time data stream. However, you realize an immediate problem upon the implementation of your software solution: your analytics engine wants to process many records of data at once, but your streaming engine wants to send individual records. How do you store this streaming data? How do you tell the analytics engine about the updates? This paper explains how to manage real-time streaming data in a batch processing analytics engine.

The problem of managing streaming data in analytics engines comes up in many industries: energy, finance, health care, and marketing to name a few. The solution described in this paper can be applied in any industry, using features common to most analytics engines. You will learn how to store and manage streaming data in such a way as to: guarantee the analytics engine has only current information, limit interruptions to data access, avoid duplication of data, and maintain a historical record of events.

.

INTRODUCTION

The purpose of this paper is to teach you how to enable real-time analysis in a high-performance analytics engine that receives frequent or real-time streams of data. This paper introduces concepts in a general way so that they may be applied in any software setting, though the example will be in SAS®.

This paper first identifies the challenges of real-time analysis and defines key concepts like batch processing engine and real-time data stream. Then it presents the features your analytics engine needs to support real-time analytics and how to implement a program that uses these features. Lastly, you'll see an example of real-time electricity load forecasting that uses these techniques.

CHALLENGE

This day in age, data analysis is often means big data analysis. It's easier than ever to collect data, inexpensive to store data, and impressive computing power is very accessible. These are all good things, but the complexity and speed that we can achieve comes with some challenges.

One aspect of model complexity is in the amount of data they process at one time. The models implemented in high-performance analytics software tend to take an aggregate, or system, view of things. They may require observations over a period of time. These models are known to consume and generate large volumes of data.

High-performance analytics is also about speed. Data must be collected, sent, and received very quickly so that decisions can be based on the most up-to-date information. This work is usually done by an event stream processing engine, such as the SAS Event Stream Processor. The event stream processing engine is connected to the data source and it streams data, ideally in-memory, to the consuming application. For the analytics engine, it's the receiving of the real-time stream that is hard.

The challenge for the analytics engine comes from the combination of complexity and speed. Complex analytics engines generally want one big static collection of data to process. This is known as batch processing. With a live data stream into the engine, there is no big static collection of data. Things are moving in and out and not in a predictable way. The engine receiving the data must intelligently incorporate this movement into its analysis, managing new records, updates, and out-of-date records. And very importantly, it must do all of this without interrupting an analysis in progress.

Also, there is often a challenge in maintaining a historical record of data so that an analysis may be repeated at a future time. When you have this constraint, you cannot simply update records, replacing the old with new. You must keep a history of events and values in order to look back.

SOLUTION

NECESSARY FEATURES OF THE ANALYTICS ENGINE

The purpose of this paper is to show you how you can support real-time analytics in an engine that operates on a big-static collection of data. In order to do this, there are several features that the analytics engine needs to have. This section covers what they are; the next section covers how you use them.

First the analytics engine needs to be able to receive real-time or near real-time data. Your engine may support a connection to a specific event stream processor for an in-memory stream; this is ideal. If your engine does not have this support built-in, you can simply provide data in files, and the files would be received frequently as data events are sent. Determining the number of events, or records, for the analytics engine to receive at once will depend on the desired frequency of updates and the speed of the analysis you perform. This will be covered in more detail the implementation section.

Second, the analytics engine needs to be able to join data in-memory for analysis. This means that you can have two or more distinct sets of data that are treated as one big collection of data by the analytics engine. While this is a fairly specific requirement of the analytics engine, it is a common feature of high-performance analytics engines. This is because big data often needs to be broken into parts for processing and storage purposes. Without the in-memory join, you would have to create giant files that are joined copies of the original input data.

Third, the analytics engine needs to be able to filter data in-memory for analysis. You need to be able to set a common attribute for each data record and tell the analytics engine to filter on this attribute as part of the analysis. The reason this must be done in memory is because the attribute that is used for the filter is assigned in-memory. That is the next requirement.

Fourth, the analytics engine needs to be able to associate in-memory an attribute with a data record. This means that the attribute is not stored in the data record and can be updated without changing the data record.

What's noticeably missing from the list above is a requirement to update data records in storage. This is intentional. This approach imposes the constraint that historical data is preserved because this is often required in practice. Also, it may be the case that your analytics engine does not support updates of data. This implementation will force the analytics engine to manage updates without changing the data that's been stored.

Another note on the requirements above is that it if your analytics engine lacks some of the features listed, you may still be able to implement real-time analysis according to the method in this paper. If your analytics engine can read its data from in-memory sources, you can implement some of the joining and filtering yourself. This paper will not cover the details of that work around.

PROCESS FOR IMPLEMENTATION

Using the features listed in the section above, you can turn your high-performance analytics engine into a real-time, or near real-time, high-performance analytics engine.

Keep in mind that this method considers a few key constraints. The first is that there are no changes to the stored data. The second is that there is no disruption any analysis in progress. The third is that copying data and processing unnecessary data is to be avoided.

Set up the real-time stream

The first step is to connect the data stream. Your analytics engine may come with support for a particular streaming engine. In this case, the data can be streamed in memory. Otherwise, you may will need to write data to disk and program your analytics engine to read data from the files.

You need to determine how many records your analytics engine will receive at once. The true real-time implementation would be to receive records as they come through the event stream. However, this may not be feasible or desirable for a few reasons. One, the records may be coming too quickly for the speed of the analytics engine. For example, the analytics engine may take two minutes to process inputs, in which case updates every two seconds would be impossible. Second, it is convenient to store data as it is received and storing data one or two records at time would result in too many files to manage. You may want to set the engine up to pool records until a certain number of records have been received or a certain amount of time has passed.

The result of the real-time stream will either be a collection of records grouped together by the analytics engine and held in memory or a set of files on disk. The next step will be to process the data.

Join the data in-memory

The records of data the analytics engine needs to process are distributed among groups of records, either separate groups in memory or separate files on disk. For analysis, the records need to be treated as one big collection of data. You need to tell the analytics engine which groups to join together and this joining of data needs to take place in memory.

Knowing what to tell the analytics engine to join requires some effort. The analytics engine needs to join all groups, or files, that contain at least one current data record. It should not include any groups that contain only out-of-date records. You will need to track what is in each group to determine the list of groups for the analytics engine to join.

If you know you need a historical record of events in order to reproduce an analysis, you may augment this data set with information about the event and the time the even occurred. Given a time in history, you can easily determine which groups would need to be joined to reproduce an analysis as of that time.

Filter the data in-memory

The groups of data joined together by the analytics engine may include out-of-date records that should not be included in analysis. This happens when updates are received erratically. You need to tell the analytics engine which records of the group should be included in the analysis. You do this by setting an attribute on each record that the engine will use as a filter. This attribute cannot be stored with the data record. This attribute, which marks a record for inclusion, will change when updates are received or when re-running a historical analysis. Some analytics engines support a concept called star schema in which attributes are linked to records rather than stored with the record. Star schema makes this very easy.

As with the list of groups to join, managing the inclusion attribute requires some effort. Here is an example of a SAS data set that sets this attribute.

EXAMPLE

The need for real-time analytics exists in every industry. This paper presents an energy and utilities industry example.

An electric utility must manage its electricity generation to serve electricity demand. This exercise, called load balancing, relies on an analytics engine to compute the current generation and consumption as well as to forecast future load. The current load analysis is a sum, but the forecast model is very complex. This paper will not go into the model itself.

As indicated in the introduction, the implementation process described in this paper is fairly generic. A number of analytics engines exist, or can be built, that have the features required. This example will be shown using the SAS High-Performance Risk analytics engine. The High-Performance Risk product has all necessary functionality out-of-the-box and is easy to use.

SET UP THE REAL-TIME STREAM

Overview

The inputs sent to the analytics engine will be the current electricity consumption of each house and business and the current generation of the plants. This example is a simplification, it leaves out inputs like weather and commodity prices.

Error! Reference source not found. is a picture of the electricity system. There are four houses and three businesses consuming electricity and two solar farms and three gas plants producing electricity.

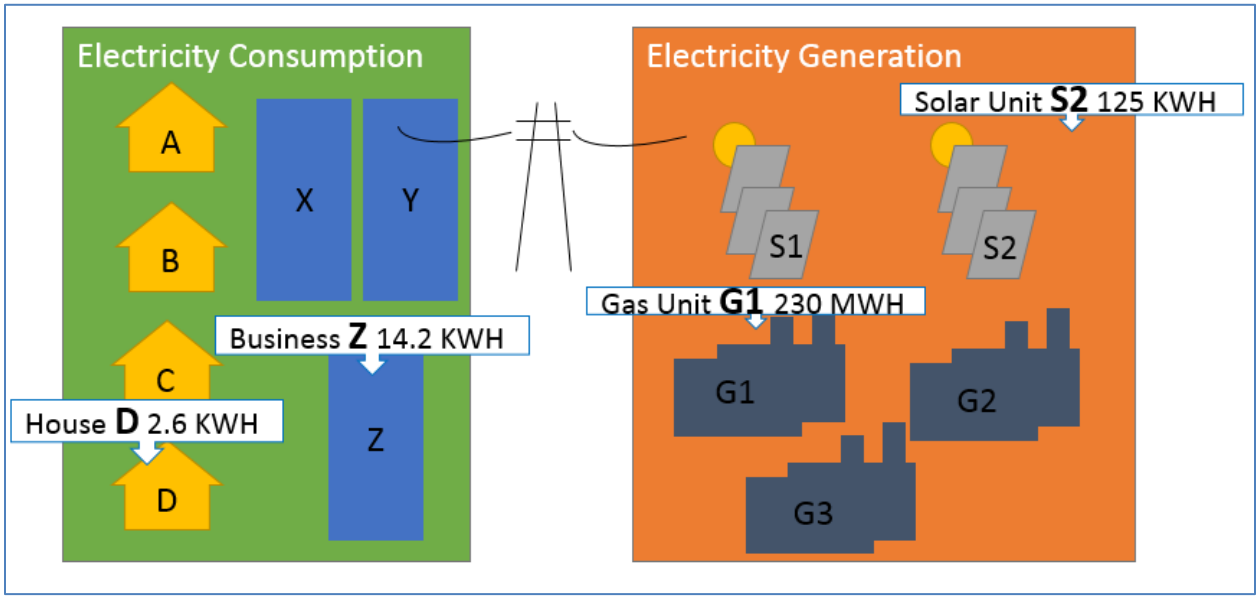

Figure 1. The Electricity System of Consumers and Producers

The houses and businesses as well as the plants are all connected to sensors, which stream data through the SAS Event Stream Processor to SAS High-Performance Risk. You will need to configure the SAS Event Stream Processor to send records with the necessary fields. This paper does not cover the ESP configuration.

From High-Performance Risk, you can choose how many records you want SAS High-Performance Risk to receive at once. In this example, SAS High-Performance Risk is configured to receive a group of three records. Each time SAS High-Performance Risk receives a group, it runs a project, which creates a risk cube, which stores the records and intermediate model results on disk.

Table 1 shows the first group records as they are streamed into SAS High-Performance Risk. The fields InstType and InstID are required for High-Performance Risk. The rest of the fields are custom attributes describing the records.

StreamID	InstType	Location	RecordID	LocationType	Role	Amount
Day1_Group1	load	House_A	House_A_Day1Group1	House	Consumer	2.1 KWH
Day1_Group1	load	House_B	House_B_Day1Group1	House	Consumer	2.4 KWH
Day1_Group1	load	House_C	House_A_Day1Group1	House	Consumer	2.7 KWH

Table 1. Records streamed from ESP to High-Performance Risk

Code

Here is the code to connect to ESP from High-Performance Risk and create a risk cube. The parameters are based on the ESP stream you configure. The disconnect_count option tells High-Performance Risk to collect three records from the data stream in a group:

```
proc High-Performance Risk
    task = runproject
    expprojlib = work
    espmode = on
    cube = "c:\temp\Electricity_Day1Group1"
    filesprefix = "local\data\risk\ElecLoadFcst_Day1Group1";
esp
    host = "my_esp_host"
    port = 1111
    positionqryname = "posqry"
    positionwindow = "poswindow"
    disconnect_count = 3;
schema
    IncludeFilter
    data = work.includeSchema
    crossclassvar recordID;
performance
    nodes = 10 nthreads = 8;
crossclassvars
    Location LocationType Role InstID;
ccfilter
    includeFlag in ("Y");
run;
quit;
```

CREATE THE INITIAL ANLAYSIS

It's easiest to think about the process starting from the very beginning, when the first data is sent to High-Performance Risk, before any updates are received from the data stream.

In the previous step, High-Performance Risk is configured to create risk cubes when a group of records is received from ESP. Each group has records for a subset of the electricity system. Once all of the groups are received, you need to join the risk cubes together to analyze the entire system.

Overview

In the initial analysis, you tell High-Performance Risk to join all of the risk cubes together. You will know the risk cubes to join together if you actively manage a ledger that identifies the cube with the most current record for each location. You can produce this ledger using output from ESP that you stream to a data set. The CurrentStream is generic in this example, in practice this would be a datetime value.

Table 2 shows the risk cubes associated with the current record of each location.

Location	CurrentStream	Cube
House_A	Day1_Update1	Electricity_Day1Group1
House_B	Day1_Update1	Electricity_Day1Group1
House_C	Day1_Update1	Electricity_Day1Group1
House_D	Day1_Update2	Electricity_Day1Group2
Business_X	Day1_Update2	Electricity_Day1Group2
Business_Y	Day1_Update2	Electricity_Day1Group2
Business_Z	Day1_Update3	Electricity_Day1Group3

Location	CurrentStream	Cube
Gas_G1	Day1_Update3	Electricity_Day1Group3
Gas_G2	Day1_Update3	Electricity_Day1Group3
Gas_G3	Day1_Update4	Electricity_Day1Group4
Solar_S1	Day1_Update4	Electricity_Day1Group4
Solar_S2	Day1_Update4	Electricity_Day1Group4

Table 2. Cube manager data set

Code

To join the risk cubes in High-Performance Risk, you run the code below with a data set listing the cubes. The data set is shown below in table 3. Here is the code to join the risk cubes:

```
proc High-Performance Risk
    task = joincubes
    cube = "c:\temp\ElectricLoadForecast"
    subcubesds = work.riskcubelist;
    ccfilter
        includeFlag in ("Y");
run;
quit;
```

Table 3 is the data set work.riskcubelist.

Cube
c:\temp\Electricity_Day1Group1
c:\temp\Electricity_Day1Group2
c:\temp\Electricity_Day1Group3
c:\temp\Electricity_Day1Group4

Table 3. List of risk cubes to join in initial anlaysis.

When High-Performance Risk is asked to analyze the joined risk cube created above, it will load data from the risk subcubes and join the records together in memory.

To analyze the initial system, you run the code below or you open the risk cube in the High-Performance Risk user interface:

```
proc High-Performance Risk
    task = query
    cube = "c:\temp\ElectricLoadForecast";
    query _tree_;
    output
          summary = work.HPRSummary;
run;
quit;
```

The output will include the current generation and consumption and the forecast generation and consumption output from the model. The actual model and analysis specifications are configured using the COMPILE and RISK procedures. That code is outside the scope of this paper.

RECEIVE UPDATES AND RUN ANALYSIS

Overview

After the initial data is received, updates will begin to stream into the analytics engine. In this example, High-Performance Risk is configured to receive updates in groups of three. Updates include new values for electricity generation or consumption, the addition of consumers or production units, and discontinued consumers or units.

In this example, the first set of updates are new consumption amounts for houses A and B and a new generation amount for solar farm S1. The updates are received by High-Performance Risk and stored in a risk cube.

Table 4 shows the updated records.

StreamID	InstType	Location	RecordID	LocationType	Role	Amount
Day1_Group5	load	House_A	House_A_Day1Group5	House	Consumer	2.1 KWH
Day1_Group5	load	House_B	House_B_Day1Group5	House	Consumer	2.4 KWH
Day1_Group5	load	Solar_S1	Solar_S1_Day1Group5	Solar	Producer	115 KWH

Table 4. First set of updates received from the data stream

Code

This is the code to create the risk cube containing the updates:

```
proc High-Performance Risk
    task = runproject
    expprojlib = work
    espmode = on
    cube = "c:\temp\Electricity_Day1Group5"
…
run;
quit;
```

Table 5 shows an updated cube manager data set

Location	CurrentStream	Cube
House_A	Day1_Update5	Electricity_Day1Group5
House_B	Day1_ Update5	Electricity_Day1Group1
House_C	Day1_Update1	Electricity_Day1Group1
House_D	Day1_Update2	Electricity_Day1Group2
Business_X	Day1_Update2	Electricity_Day1Group2
Business_Y	Day1_Update2	Electricity_Day1Group2
Business_Z	Day1_Update3	Electricity_Day1Group3
Gas_G1	Day1_Update3	Electricity_Day1Group3
Gas_G2	Day1_Update3	Electricity_Day1Group3
Gas_G3	Day1_Update4	Electricity_Day1Group4
Solar_S1	Day1_Update5	Electricity_Day1Group5
Solar_S2	Day1_Update4	Electricity_Day1Group4

Table 5. Cube manager data set after fist set of updates

It is recommended that in addition to Table 5, you create a data set that contains the names of the events and the time stamp of the events. This provides a way to look back in history to reproduce analysis as of a certain date and time. Note, tables 4 and 5 should be kept separately as shown here because table 4 will be used regularly and should only be as long as the number of locations. Table 5 will grow very large and will need regular archiving, and it will be used seldomly.

Table 6 shows a full historical view of events. The StreamID represents the time stamp of the event.

Location	StreamID	Cube	Event
House_A	Day1_Update5	Electricity_Day1Group5	Update
House_A	Day1_Update1	Electricity_Day1Group1	Create
House_B	Day1_Update5	Electricity_Day1Group1	Update
House_B	Day1_Update1	Electricity_Day1Group1	Create
House_C	Day1_Update1	Electricity_Day1Group1	Create
House_D	Day1_Update2	Electricity_Day1Group2	Create
Business_X	Day1_Update2	Electricity_Day1Group2	Create
Business_Y	Day1_Update2	Electricity_Day1Group2	Create
Business_Z	Day1_Update3	Electricity_Day1Group3	Create
Gas_G1	Day1_Update3	Electricity_Day1Group3	Create
Gas_G2	Day1_Update3	Electricity_Day1Group3	Create
Gas_G3	Day1_Update4	Electricity_Day1Group4	Create
Solar_S1	Day1_Update5	Electricity_Day1Group5	Update
Solar_S1	Day1_Update4	Electricity_Day1Group4	Create
Solar_S2	Day1_Update4	Electricity_Day1Group4	Create

Table 6. Cube manager data set after fist set of updates

To incorporate the updates into the analysis, you create a new join risk cube in High-Performance Risk that is made up of the unique risk cubes from your updated risk cube manager data set (see table 5).

Table 7 shows the new risk subcube list you pass to High-Performance Risk for the join.

Cube
c:\temp\Electricity_Day1Group1
c:\temp\Electricity_Day1Group2
c:\temp\Electricity_Day1Group3
c:\temp\Electricity_Day1Group4
c:\temp\Electricity_Day1Group5

Table 7. List of risk cubes to join in initial analysis.

Here is the code to incorporate the updates. Note the CLEAN task, which is necessary to update the join cube:

```
proc High-Performance Risk
    task = clean
    cube = "c:\temp\ElectricLoadForecast";
run;
quit;

proc High-Performance Risk
    task = joincubes
    cube = "c:\temp\ElectricLoadForecast"
    subcubesds = work.riskcubelist;
```

```
    schema
        IncludeFilter
        data = work.includeSchema
        crossclassvar recordID;
ccfilter
    includeFlag in ("Y");
run;
quit;
```

When this code finishes, the joined risk cube will contain two observations for houses A and B and two observations for solar farm S1. SAS High-Performance Risk must be told to discard the old observations when it computes the analysis. The schema statement and ccfilter statement in the code above are used for that.

SAS High-Performance Risk has the capability to filter records out of the analysis at run-time based on any attribute. In this example, the attribute will be called includeFlag and it will take value "Y" to indicate the record is to be included and "N" to indicate the record is to be excluded.

The attribute will be assigned in a separate SAS data set, known to High-Performance Risk as a schema data set, and will be linked to the records based on recordID_time, which captures the location (ie which house, business, or plant) in the record and the time of the record. The schema data set can be updated any time and is read by High-Performance Risk at the time the analysis is run.

Table 8 shows the schema data set after the first set of updates is received. This is the data set work.includeSchema that is specified on creation of the joined risk cube.

RecordID	**IncludeFlag**
House_A_Day1Group5	Y
House_A_Day1Group1	N
House_B_Day1Group5	Y
House_B_Day1Group1	N
House_C_Day1Group1	Y
House_D_Day1Group2	Y
Business_X_Day1Group2	Y
Business_Y_Day1Group2	Y
Business_Z_Day1Group3	Y
Gas_G1_Day1Group3	Y
Gas_G2_Day1Group3	Y
Gas_G3_Day1Group4	Y
Solar_S1_Day1Group5	Y
Solar_S1_Day1Group4	N
Solar_S2_Day1Group4	Y

Table 8. Schema data set to mark records for inclusion after first set of updates

Now the out-of-date records are marked for exclusion and the updates are marked for inclusion. The CCFILTER statement that you specify when you create the joined risk cube tells High-Performance Risk to apply a filter that only includes records where includeFlag is set to "Y."

The results of the analysis on the updated risk cube will incorporate the updates to houses A and B and solar farm S1.

A quick note on interruption of current processes: the way this is implemented in High-Performance Risk, there is no interruption of analyses in progress. The joined risk cube, described earlier as a set of instructions, can be updated without bothering existing sessions. All sessions use the set of instructions

to do an initial load of data into memory and start the analysis and then they're done with it. The same thing is true of the schema data set.

CONTINUING THE UPDATES

Updates will continue to stream into High-Performance Risk and the steps above are repeated with each update. The number of risk cubes joined each time will grow and shrink in the process of updating existing records. In this paper's example, the number of risk cubes grew from four to five when the first update was received. Once an update is received for House C, the first risk cube will be entirely replaced and it will no longer be in the list of risk cubes to join. For best performance, you should try to keep the number of risk cubes joined under 1,000 if possible.

CONCLUSION

It is possible to configure a high-performance analytics engine to respond to real-time data streams even if the engine is designed to process data in single batches. Further, it is possible to do this without interrupting analysis in progress and without changing any stored data.

ACKNOWLEDGMENTS

Stacey Christian and Don McAllister reviewed this paper and provided valuable feedback.

RECOMMENDED READING

- *SAS® Risk Dimensions and SAS® High-Performance Risk Procedures Guide*

CONTACT INFORMATION

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

Katherine Taylor
Katherine.taylor@sas.com