A PRACTICAL JOURNAL OF z/OS TUNING AND MEASUREMENT ADVICE

Cheryl Watson's

REPRINT

Tuning Letter

This document is a reprint of an article from *Cheryl Watson's Tuning Letter 2016 No .3*. It is the first of a series of articles that will document our experiences with using Spark on z/OS to process SMF data. If you are not familiar with the Tuning Letter, you can see the Table of Contents for this and all other issues on our website at http://watsonwalker.com/publications/tuningletter/table-of-contents/.

Spark for z/OS and SMF - Part 1

In *Tuning Letter 2016 No. 1*, we had an article about the new streaming capability in SMF ('SMF Streaming'), and we said that we would follow up with an article about how this capability can be combined with Apache Spark to run analytics against SMF data.

Since then, we have been working with IBM and Rocket Software to better understand the capabilities of the Spark package (the official name is 'IBM z/OS Platform for Apache Spark') in general, and specifically in relation to its use with SMF data. The implementation of a full Spark environment is not trivial (at least, not for an old-timer like me!), so we believe that taking an incremental approach will help you get up and running faster, and then you can move on to more complex processing as your familiarity with Spark grows.

As a result, our plan is to use *this* article to give you the 'big picture' overview of Spark on z/OS, and to discuss how you can, relatively easily, create a test environment where you can run SQL queries against SMF data residing in sequential data sets. Our next installment will cover the next logical step - using Spark to produce traditional reports from large volumes of SMF data using Java or Scala and SQL. And the final installment will delve into unleashing the real potential of SMF data; using analytics with your SMF data, and even including real time SMF data.

In this issue, we'll show you how to access sequential SMF data sets using SQL

We want to especially thank **Erin Farr, Mike Gildein**, **Neil Shah**, and **Elpida Tzortzatos** from IBM, and **Steve Bowen**, **Jamie Parsaie, Gregg Willhoit** and their colleagues from Rocket Software for their endless patience with all my questions. There is no way we could have created these articles without their enthusiasm and help.

If you are as new to analytics as I am and a Tuning Letter subscriber, please take a few minutes to review 'What Is...Analytics?' on page 18 of the 2016 No 3 issue. It is a light-hearted introduction to analytics, and it might help get you thinking about how you could combine the power of analytics with the wealth of information that is lurking in your SMF records.

Introduction

A good place to start would be to describe precisely what the IBM z/OS Platform for Apache Spark (product 5655-AAB) consists of. It was formally announced on March 22nd, 2016, in IBM US announcement number 216-067.

The product that you can 'purchase' from IBM (it is actually a no-charge product) consists of two main components:

- An open source analytics infrastructure called Apache Spark[1].
- A subset of Rocket Software's DVS (Data Virtualization Service) product. When delivered as part of the Spark package, this function is called MDSS (Mainframe Data Service for Apache Spark).

To grossly over-simplify, MDSS is responsible for gathering the data that the query or analysis will be run against. And Spark provides the infrastructure and the 'smarts' to run the query or analysis as quickly and efficiently as possible.

I think it will help you visualize the overall offering if you bear in mind that Spark and MDSS are effectively two separate products, delivered with a large amount of integration to glue them together. There might be a single product number and a single SMP/E install, but the administration, setup, management, and most of the documentation are still handled as if they were two products. In fact, it *is* possible to download the Spark part on its own (it *is* open source, after all), however, that option doesn't deliver the ability to

[1] To avoid confusion, when referring to the 5655-AAB product (which includes MDSS and Apache Spark), we will call it the 'Spark package'. If we are referring just to the Apache Spark component of that product, we will simply call it 'Spark'.

access z/OS data other than IMS or DB2 data. It also does not provide the Service and Support option that is available for the IBM z/OS Platform for Apache Spark product.

Before we go on, and to ensure that we keep your attention, let's get the all-important stuff out of the way first - money.

Don't you just love that word - 'free'

- The IBM z/OS Platform for Apache Spark is **free**. No MLC or Value Units or One Time Charge. The only thing you pay for is Service & Support, and that is optional.
- Spark itself, *and* all your programs that run under Spark, are **100% zIIP-eligible**, so Spark can potentially be run with nearly zero impact on the monthly license charges for all your z/OS-based products.
- The MDSS processing that retrieves the data for Spark is also **nearly 100% zIIP-eligible**. The only MDSS impact on general purpose CPs is if MDSS retrieves data from DB2, in which case some of the DB2 processing is not zIIP-eligible (that is under the control of DB2).
- IBM is offering a discount on memory that is purchased specifically for use with Spark (the rumor mill has it that it is a *very* attractive discount - but you didn't hear that from me).
- IBM is offering a discount on zIIPs that are purchased specifically for use with Spark.

This adds up to what must be the lowest cost work you will ever run on z/OS.

So, now that you have kept your accountants happy, let's go and look at Spark and MDSS in a little more detail.

What Is Spark?

Spark is an open source computing framework, specifically designed for clustered, very high performance analytics - you can view its home page at http://spark.apache.org/. It was originally developed at University of California at Berkeley in 2009. It is now one of the largest open source projects, and also one of the fastest growing. It is used by companies such as Yahoo, Netflix, Facebook, and eBay. And now Watson & Walker. In other words, all the major trans-galactic computing giants ☺.

Something to bear in mind if you are interested in using Spark with SMF data is that Spark is a *framework* - it is *not* an application or a solution. You might liken it to CICS, where CICS provides an infrastructure to run the transactions that run your business, but it requires someone (you, or an application vendor) to write all the transactions. Similarly, Spark provides the ability to run queries against data that has been loaded into one of its Resilient Distributed Datasets (RDDs) (these are effectively in-memory databases, although, strictly speaking, they can be defined to use disk as well), and to use massive parallelism to complete complex analyses in tiny amounts of time (think in terms of seconds or less). But *you*, or some vendor, need to create the queries and analytics programs that will run under Spark.[2]

Coming soon - sample SMF queries on the Watson & Walker website

Because it is still so new, I am not aware of any applications for Spark that will process SMF data yet. However, the folks at Rocket is actively looking at providing a number of sample programs and supporting an online community where people can pull down samples, and hopefully contribute some of their own. Because Cheryl is universally acknowledged as Ms. SMF, we will be hosting these samples on our website (www.watsonwalker.com). You know how much Cheryl loves SMF, and I heard that she was spotted skulking around the Java books in her local Barnes and Noble recently, so who knows that you might find out there. All of this was only agreed in the last few days, so it will take us a little while to get this set up and running - we will update Cheryl's List blog when the site is ready to go. We hope that we can count on our readers to play an active part - remember, we all benefit when we all share.

Spark, and all the queries that run under it, can run nearly entirely on zIIPs.

Spark itself is written in a programming language called Scala, and it can run user programs written in Scala, Java, or Python[3]. Because Scala and Python are based on Java, the Spark executors, and all the user programs that use Spark, are zIIP-eligible. On z/OS, a Spark cluster (or clusters) runs in one or more JVMs under Unix System Services. Spark on other platforms supports the ability to have multi-node clusters that are spread over multiple servers, however, this capability is not available on z/OS yet. We don't believe that most customers will actually *need* this capability when running on z/OS; at least, not for the time being[4].

I think it is reasonable to say that IBM's primary intended market for Spark is in business analytics; things like fraud detection, analyzing customer buying patterns, determining the most profitable price for airplane seats, identifying the best customers for loan offers, and so on. The fact that they have identified SMF data as one of the early uses of Spark is a bonus for us.

IBM believes that z/OS is particularly well suited to very high performance analytics because z Systems CPCs have some of the highest speed chips in the industry, huge I/O bandwidths, oceans of data (data *lakes* are for those little distributed systems ☺), and because the very large memory available on z Systems CPCs eliminates the need to partition data across multiple servers.

One of the capabilities that the Spark package delivers is the ability to pull data from many data sources (including those on other platforms) into Spark running on z/OS. This could result in the somewhat unexpected scenario of data from z/OS and many other platforms all being sucked *into* Spark on z/OS in order to run very high performance analytics against them. While this appears to be feasible (and technically very attractive in some

[2] Watson & Walker is working with Rocket Software and IBM to create and host an ecosystem around Spark and SMF, whereby customers could contribute queries or analysis programs that they feel would be of interest to other Spark/SMF customers. We plan to seed that website with some sample programs to help customers get started.

[3] Spark also supports the popular R programming language, but R is not yet available on z/OS (Rocket Software is working on a port).

[4] According to http://www.infoworld.com/article/3014440/big-data/five-things-you-need-to-know-about-hadoop-v-apache-spark.html, 'Spark can be as much as 10 times faster than MapReduce (Hadoop) for batch processing and up to 100 times faster for in-memory analytics.'

scenarios), sadly, prevailing market winds might make this politically unacceptable. Still, if this capability turns out to be even partly as successful as IBM hopes, I can see it bringing a wry grin to the face of many mainframers.

But to return to the question of Spark and SMF. As we said, Spark is an infrastructure that can be used to process many types of data. But Spark, in and of itself, knows nothing about SMF data. Remember that it is open source, so it came from a land that never heard of SMF records. What makes Spark particularly interesting to us in Watson & Walker (and to anyone that works with SMF data) is its packaging with Rocket Software's MDSS.

What Is MDSS?

The other half of the Spark package is MDSS (Mainframe Data Service for Apache Spark). MDSS is a subset of Rocket Software's chargeable Data Virtualization Service (DVS) product. If purchased as a standalone product, DVS allows you to take many data set types, map the contents, process them with SQL as if they were DB2 tables, and save the results in one of many formats.

Because MDSS is provided for free as part of the Spark package, it is understandable that the full functionality of DVS is not included. For example, MDSS is primarily intended for the output to be sent to Spark for processing, rather than to an output file. However, the mapping of record layouts to SQL definitions is portable back and forth between MDSS and DVS, and other aspects of MDSS will be familiar to anyone that has worked with DVS.

MDSS provides the ability to map record layouts to virtual tables. For example, the fields in an SMF type 74 subtype 4 record might be mapped to a virtual table called `SMF_07404`. Regardless of where that record physically resides (it could even be on tape), you could process it in Spark using SQL statements just as if it was in a real DB2 table. This means you can use Select statements, Joins, Unions, Order by, WHERE clauses (to let you filter the set of records that you want to work with), and so on. You have all the flexibility and power that you would have if the SMF data was stored in DB2, but without the CPU and disk space cost of having to load it into DB2. You also don't have the unenviable task of creating DB2 DDL statements to match the SMF fields because MDSS comes with mappings for most of the popular SMF record types (more on that in a minute).

Further, using this example of doing a Select against the RMF 74.4 table, assume that the 74.4 records are held in the same data set as all your other SMF records. You obviously don't want *all* the fields from *all* the SMF records getting passed back to Spark. So, in this example, MDSS will save only the 74.4 records in its local buffers as it reads through the SMF data sets. Then, when it is passing the data back to Spark, it extracts out just the specific *fields* that were specified on the Select statement. This means that you could start with a multi-GB data set containing *all* SMF record types, but end up with a relatively small volume of data being sent back to Spark.

Because this series of articles is specifically about using the IBM z/OS Platform for Apache Spark with SMF records, we will focus on how MDSS handles SMF data.

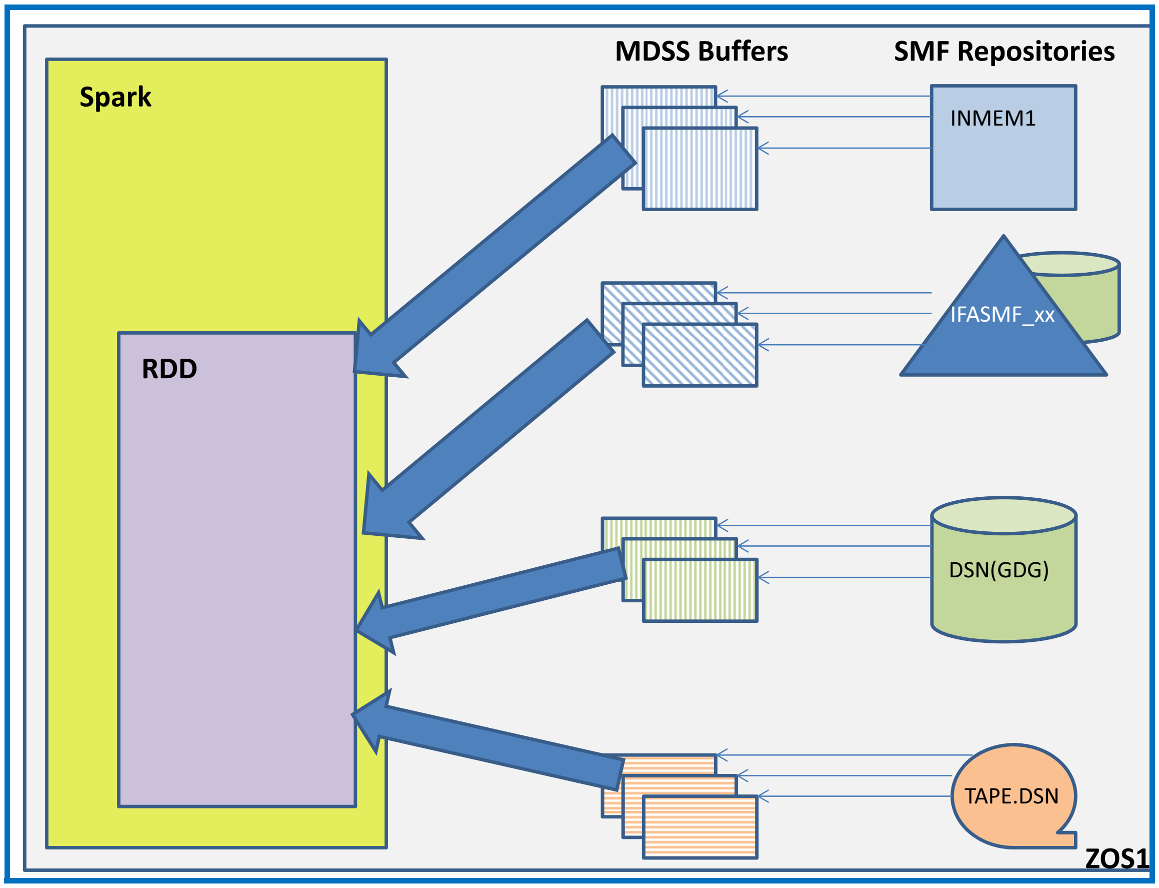
MDSS comes packaged with mappings for many of the popular SMF record types (0, 6, 14, 15, 23, 26, 30, 42, 60, 70-79, 80, 81, 83, 88, 89, 100, 101, 102, 110, 113, 115, 116, 118, and 119). At the time of writing, they have over 1500 virtual tables and views covering these record types, and that list is constantly being expanded. We have found Rocket to be very responsive to requests for mappings of additional record types.

MDSS can retrieve SMF data from:

- Sequential data sets on DASD.
- SMF Log streams[5].
- Data sets on tape.
- One or more of the new in-memory buffers using the new Streaming SMF support[6].

This is illustrated in Figure 1.

Figure 1 - Retrieving SMF Data into the RDD



The MDSS buffers in the center of the figure represent multiple sets of buffers in MDSS. Multiple buffers allow MDSS to read different portions of the SMF repository (a sequential data set, for example) in parallel. They also enable high volume transfers of data from the buffers over to the Spark RDDs.

The RDD resides in the system that the query was initiated on. A representation of a simple Spark environment is shown in Figure 2.

[5] It also supports log blocks that have been compressed using zEDC.
[6] This support was added to Spark by APAR PI59392, and to SMF by APAR OA49263.

Figure 2 - Simple Spark Environment

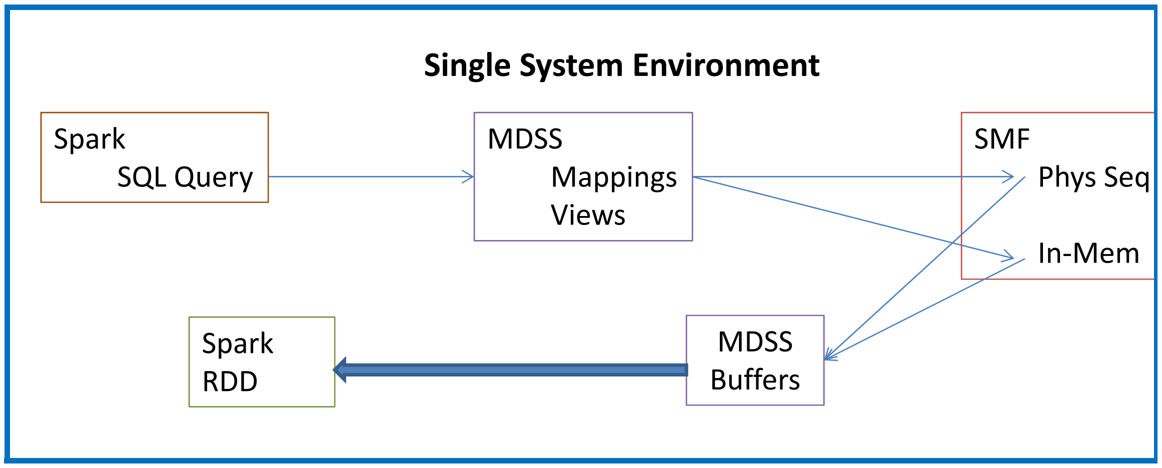


Figure 2 shows a single system. In this example, SMF data resides in a combination of sequential data sets and SMF in-memory buffers (SMF record types that are using the Streaming support). A query is initiated under Spark. That sends off a request to MDSS for the required SMF data. MDSS retrieves the data (using as much parallelism as you allow), into a set of buffers that are dedicated to that connection, and then passes the data back to Spark, which saves it in an RDD.

It is likely that many customers that are interested in using Spark to process SMF data will have more than one system. If you merge the SMF data from all systems into a single set of data sets, then the MDSS that is local to Spark would retrieve all the requested data.

However, consider the situation where you have four systems, and your SMF archive data sets are structured so that each data set contains data from just one system. In that scenario, you might want (or need) MDSS on each system to retrieve the SMF data from its own set of SMF archive data sets. This would especially be the case if the archive data sets are not accessible from all the systems in the sysplex.

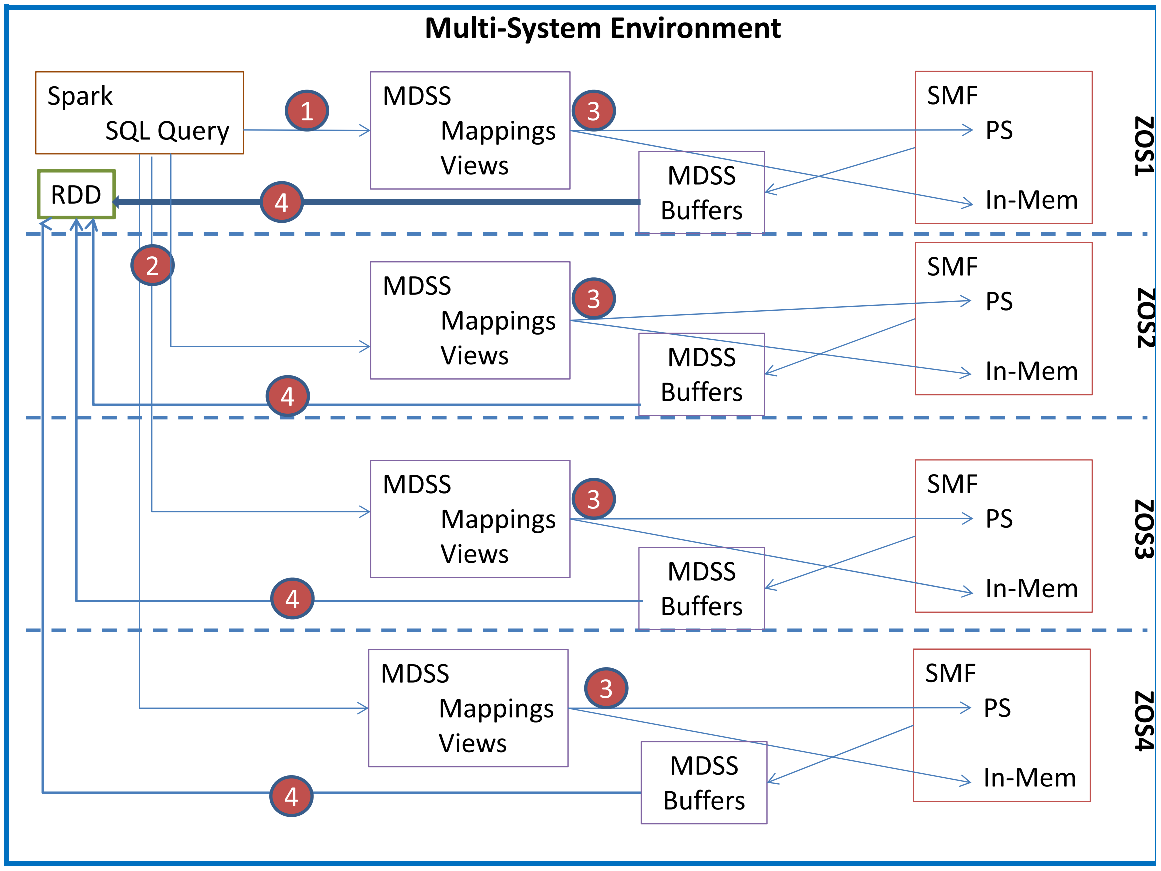
In this case, you have two choices for accessing the data on the other systems:

◆ Spark can send its requests to the local MDSS, which then passes the requests for the remote data to MDSS on those systems. The remote MDSS' send the data back to the local MDSS which then passes it back to Spark.

◆ The other, preferred, option is that Spark sends its requests directly to the remote systems, which then send the data directly back to Spark.

An example of the second configuration is shown in Figure 3 on page 8. The systems do not all have to be in the same sysplex. TCP/IP is used to communicate between Spark and the MDSS started tasks in the other systems - this means that Spark can benefit from new, high performance, network technology such as SMC-R and SMC-D (for LPARs that reside on the same CPC). Also, MDSS can use zEDC (if present on both CPCs) to reduce the size of the data blocks being sent between systems.

Figure 3 - Spark in a Multi-System Environment



In the diagram above, the steps are as follows:

1. Spark on system ZOS1 sends the request for data that resides on ZOS1 to MDSS on zOS1.
2. Spark also sends its requests directly to the MDSS subsystems on each of the other systems where the required data resides.
3. MDSS on each system starts retrieving the data from the SMF data sets on that system into the local MDSS buffers.
4. MDSS on each system applies the filters from the Select statement and sends the data back to Spark on system ZOS1.

MDSS is primarily intended to work with Spark. Spark enables high performance analysis of data, and MDSS feeds data to Spark from nearly every type of z/OS data set (sequential, partitioned, VSAM, IMS, DB2, and ADABAS). It also supports the ability to process all those data sources with SQL.

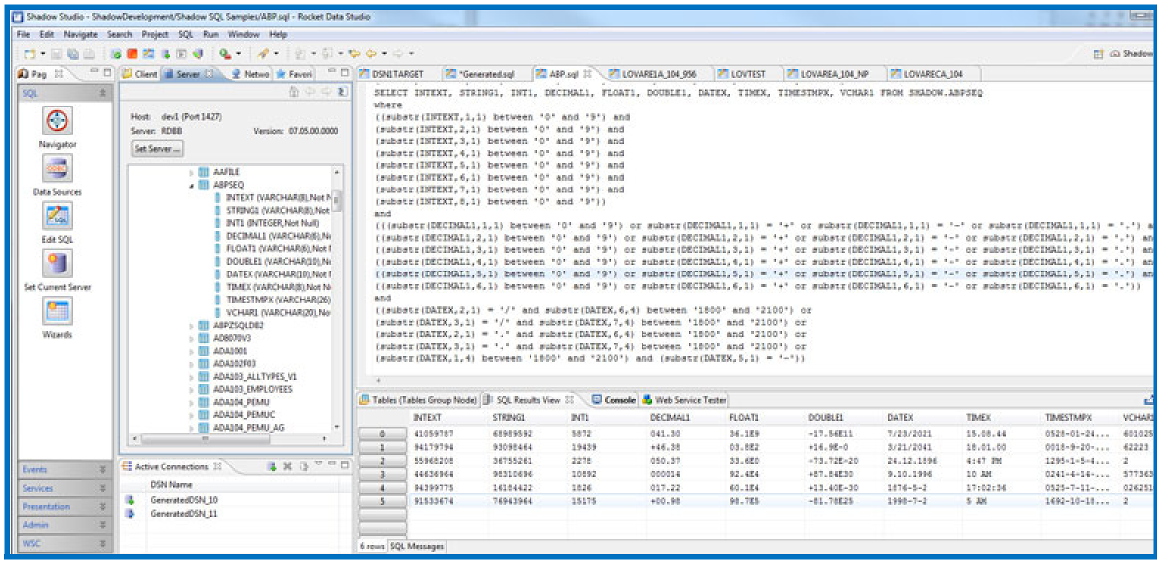
Other Components

The other two components that are included in the Spark package are the Data Service Studio and a JDBC driver. These are actually part of the MDSS half of the Spark package.

Data Service Studio

The Data Service Studio is provided with MDSS (that is, it is not part of the open source Apache Spark). The Studio is an Eclipse-based workstation tool. It is primarily provided as a tool to help you create and test queries that will eventually be included in Java, Scala, or Python programs. You can use it to get a list of all the virtual tables that are provided with MDSS, as well as a list of the columns in each table (note that the column names for the MDSS-provided SMF virtual tables are nearly always the same as the SMF field that the data in that column is retrieved from). A sample screenshot from the Studio is shown in Figure 4. In addition to providing a facility to test your SQL statements, the Studio is also used to define new virtual tables to MDSS.

Figure 4 - Data Service Studio



```
SELECT INTEXT, STRING1, INT1, DECIMAL1, FLOAT1, DOUBLE1, DATEX, TIMEX, TIMESTMPX, VCHAR1 FROM SHADOW.ABPSEQ
where
((substr(INTEXT,1,1) between '0' and '9') and
(substr(INTEXT,2,1) between '0' and '9') and
(substr(INTEXT,3,1) between '0' and '9') and
(substr(INTEXT,4,1) between '0' and '9') and
(substr(INTEXT,5,1) between '0' and '9') and
(substr(INTEXT,6,1) between '0' and '9') and
(substr(INTEXT,7,1) between '0' and '9') and
(substr(INTEXT,8,1) between '0' and '9'))
and
(((substr(DECIMAL1,1,1) between '0' and '9') or substr(DECIMAL1,1,1) = '+' or substr(DECIMAL1,1,1) = '-' or substr(DECIMAL1,1,1) = '.') a
((substr(DECIMAL1,2,1) between '0' and '9') or substr(DECIMAL1,2,1) = '+' or substr(DECIMAL1,2,1) = '-' or substr(DECIMAL1,2,1) = '.') an
((substr(DECIMAL1,3,1) between '0' and '9') or substr(DECIMAL1,3,1) = '+' or substr(DECIMAL1,3,1) = '-' or substr(DECIMAL1,3,1) = '.') an
((substr(DECIMAL1,4,1) between '0' and '9') or substr(DECIMAL1,4,1) = '+' or substr(DECIMAL1,4,1) = '-' or substr(DECIMAL1,4,1) = '.') an
((substr(DECIMAL1,5,1) between '0' and '9') or substr(DECIMAL1,5,1) = '+' or substr(DECIMAL1,5,1) = '-' or substr(DECIMAL1,5,1) = '.') an
((substr(DECIMAL1,6,1) between '0' and '9') or substr(DECIMAL1,6,1) = '+' or substr(DECIMAL1,6,1) = '-' or substr(DECIMAL1,6,1) = '.'))
and
((substr(DATEX,2,1) = '/' and substr(DATEX,6,4) between '1800' and '2100') or
(substr(DATEX,3,1) = '/' and substr(DATEX,7,4) between '1800' and '2100') or
(substr(DATEX,2,1) = '.' and substr(DATEX,6,4) between '1800' and '2100') or
(substr(DATEX,3,1) = '.' and substr(DATEX,7,4) between '1800' and '2100') or
(substr(DATEX,1,4) between '1800' and '2100') and (substr(DATEX,5,1) = '-'))
```

	INTEXT	STRING1	INT1	DECIMAL1	FLOAT1	DOUBLE1	DATEX	TIMEX	TIMESTMPX	VCHAR1
0	41059767	68989592	5872	041.30	36.1E9	-17.56E11	7/23/2021	15.08.44	0528-01-24...	601025
1	94179794	93098464	19439	+46.38	03.8E2	+16.9E-0	3/21/2041	18.01.00	0018-9-20-...	62223
2	55968208	36755261	2278	050.37	33.6E0	-73.72E-20	24.12.1896	4:47 PM	1295-1-5-4...	2
3	44636964	98310696	10892	000014	92.4E4	+87.84E30	9.10.1996	10 AM	0241-4-16-...	577363
4	94399775	16184422	1826	017.22	60.1E4	+13.40E-30	1876-5-2	17:02:36	0525-7-11-...	026251
5	91533674	76943964	15175	+00.98	98.7E5	-81.78E25	1998-7-2	5 AM	1692-10-18...	2

The Data Service Studio is provided as a member in one of the MDSS data sets (SAZKBIN). You download the member to your PC and perform the install from there.

JDBC Drivers

Spark and the Data Service Studio both use JDBC drivers to communicate with MDSS. Spark can also communicate directly with IMS and DB2 via JDBC. The JDBC driver for MDSS is a Type 4 driver that is written in Java and implements the network protocol for the IBM z/OS Mainframe Data Service for Apache Spark.

Getting Started with the Spark Package

Remember that the objective of this first article is to provide an overview of the Spark package, and then show how you can quickly get to a point where you can assess the benefits of being able to access SMF data in sequential data sets using SQL. Now that you have a picture of how the various components relate to each other, let's look at what you need to do to get that SQL access working.

For our example, we are going to use the Data Service Studio to generate and run SQL queries against SMF data sets. This does not provide any of the performance benefits of

Spark, and you can only process SQL statements. Real world SMF processing (similar to the type of processing you do against your SMF data today) would require a program around the SQL statements, but that is not supported by the Data Service Studio. Remember that our objective is to get you up and running and able to test your SQL statements as quickly as possible. We also want to give you an opportunity to assess the value of SQL access to most of your SMF data.

The environment we used for our testing was a 4-core Dell PC running z/OS 2.2 under zPDT - about as far from a production environment as you could possibly get. Despite that, the performance was more than acceptable. We were not able to test support for certain capabilities, such as zEDC support, but other than that, it functioned perfectly.

MDSS

The Spark package includes both the Spark code and MDSS. When you run an SMP/E install, it installs both products[7]. However, in the remainder of this article we are going to focus on MDSS. It is not necessary to perform any customization for Spark, or even to start Spark, in order to be able to use the functions we will discuss here.

TIP: Review the IBM Redbook SG24-8235, *Apache Spark Implementation on IBM z/OS,* the product installation manual SC27-8449, *IBM z/OS Platform for Apache Spark Installation and Customization Guide*, and TechDoc WP102609 *Installing IBM z/OS Platform for Apache Spark* before you start the installation.

Happily, it is possible to get MDSS up and running with minimal effort and minimal customization. Considering that MDSS has over 1000 parameters, this is probably a good thing! The installation steps for MDSS are described in Section 3.2 of the Spark Implementation Redbook referenced above. We recommend that you keep things as simple as possible at this point and do not include DB2 or IMS support - you can always add them later.

Also, we want to stress that you most definitely *will* want to customize the MDSS parameters before you move it to a production environment[8]. But most, if not all, of the default values should be OK for our testing purposes.

MDSS consists of a single started task - the default name is AZKS. It also has a powerful ISPF interface that is required - there are some aspects of MDSS management that can't be performed without the ISPF interface.

When you have completed the necessary customization, it is time to start MDSS, On our little zPDT system it took a couple of minutes to finish initializing. The important thing is to watch for the following message, indicating that it has finished initializing:

```
AZK4273H OE sockets Outbound SSL connection support is being
activated
```

[7] For information about the prerequisites for the Spark package, see 'Prerequisites' on page 31.
[8] We hope to provide guidance for customizing the MDSS parms in a future issue, after we have more experience, and more feedback from customers using it in production.

You can use a batch job to ensure that you can communicate with MDSS and that you can access data sets. But before you do that, you need to define the test data set to MDSS. To do that, follow these steps:

- ◆ In ISPF, start the MDSS interface using this command: ex 'user.clist(azk)' 'SUB(AZKS)'.

  A sample AZK exec is delivered in AZK.SAZKEXEC.
- ◆ Enter ‘E.2’ to customize the rules members.
- ◆ Hit Enter on the following screen,
- ◆ Type an ‘S’ beside ‘VTB’ and hit Enter.

  VTB stands for Virtual Tables. The members that are listed contain the rules that will map the virtual table names to data set names. For now, we will just customize the member that is used for the provided SMF virtual tables and views.
- ◆ Type an ‘E’ beside AZKSMFT2 and hit Enter. This enables the rule.
- ◆ F3 back to the primary MDSS menu.

Now we need to create an MDSS Global Variable that will map the virtual table name to the data set that contains your test SMF data. You do that by:

- ◆ Go to the ‘Display Global Variables’ screen (Option E.1)
- ◆ Change the Global Prefix to ‘GLOBAL2’ and selecting subnode ‘SMFTBL2’.
- ◆ You can add new variables with the ‘Show variablename variablevalue’ command. For example, to add a variable that will map the SMF_07001 table to a data set called SMF.DUMP70, you would enter "S SMF_07001 SMF.DUMP70".

TIP: Be careful when defining the data set name. You must enter the name in upper case. If you specify the correct name, but in lower or mixed case, the allocation will fail.

Make sure that the data set that you point to in your new variable includes some type 70 records because those are the ones we will use for our test.

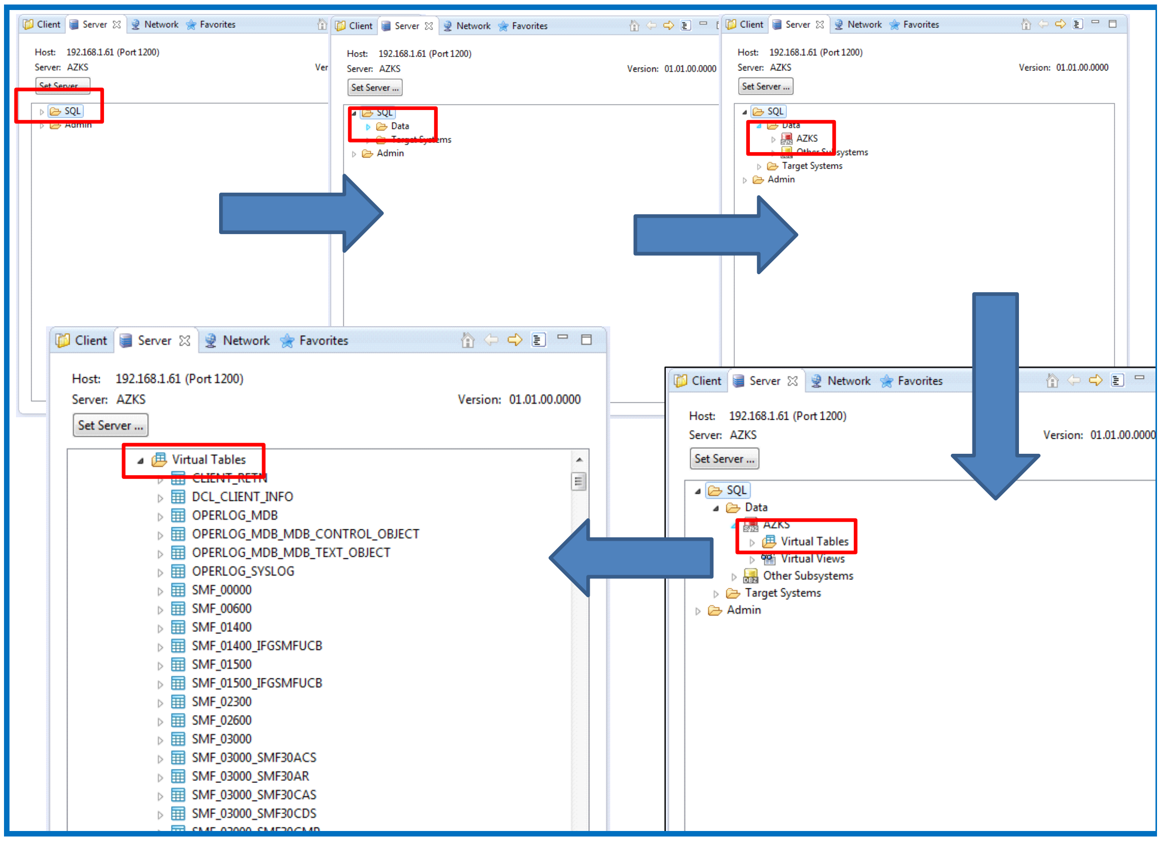
Now we are ready to test MDSS using the batch job referenced above. Copy the AZKIVVS1 member in the AZK.SAZKCNTL data set to your own data set. Then update the JOB card, update the HLQ on the SET statements, delete the first step (the IKJEFT01 one), and replace the ‘SELECT * FROM STAFFVS’ statement with ‘SELECT * FROM SMF_07001’, and submit the job. If everything has been set up correctly, the job should end in a few seconds with a return code 0, and the FMT output file should contain the contents of the type 70 records in your test data set. Congratulations! You are now an SMF master, able to extract whatever fields you want from SMF records with a simple SQL SELECT statement.

The next step is to download the Data Services Studio from the AZKBIN1 member of the AZK.SAZKBIN data set. You also need to download the JDBC driver from the AZKBIN2 member of that same data set. The ‘installation’ of the Data Studio and the JDBC driver takes about 5 minutes - just follow the step-by-step instructions in Section 3.3 of the Spark Implementation Redbook.

To start the Data Studio, run the launch.bat file in the Studio folder. When the Data Studio starts, click the ‘Set Server’ button on the Server tab. Enter the IP name or address of the system where MDSS is running, the port number (the default port number is 1200) and a userid and password that has access to your test data set. Now click OK. The studio will now connect to MDSS.

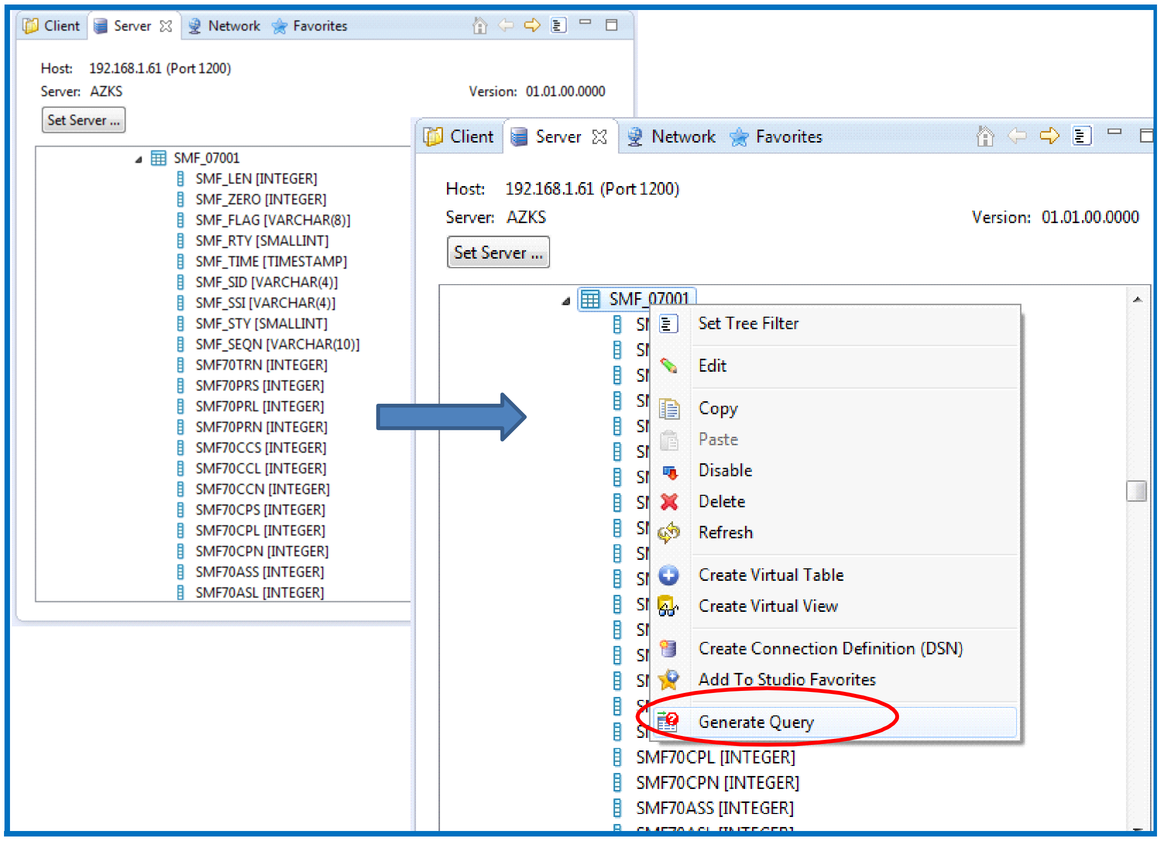
In the Server Tab, there should be two folders: SQL and Admin. Click on the twistie beside SQL. You should now see two more folders: Data and Target Systems. Click on the twistie beside Data, and you should be presented with two icons - one for your MDSS subsystem, and one for Other Subsystems. Click the twistie beside the icon for your MDSS subsystem. You will now be presented with a list of all the virtual tables and virtual views that are defined to that MDSS subsystem. All of these steps are shown in Figure 5.

Figure 5 - Displaying Virtual Tables Defined to MDSS



The screen in the bottom left of Figure 5 shows the virtual tables that are defined to the MDSS subsystem that you are connected to. In this example, you can see a number of SMF tables. And if you look at the scroll bar on the right of that screen, you can see that there are many more tables if you scroll down. To see the fields in a virtual table, click on the twistie beside a table name.

Figure 6 - Displaying the Columns in a Virtual Table

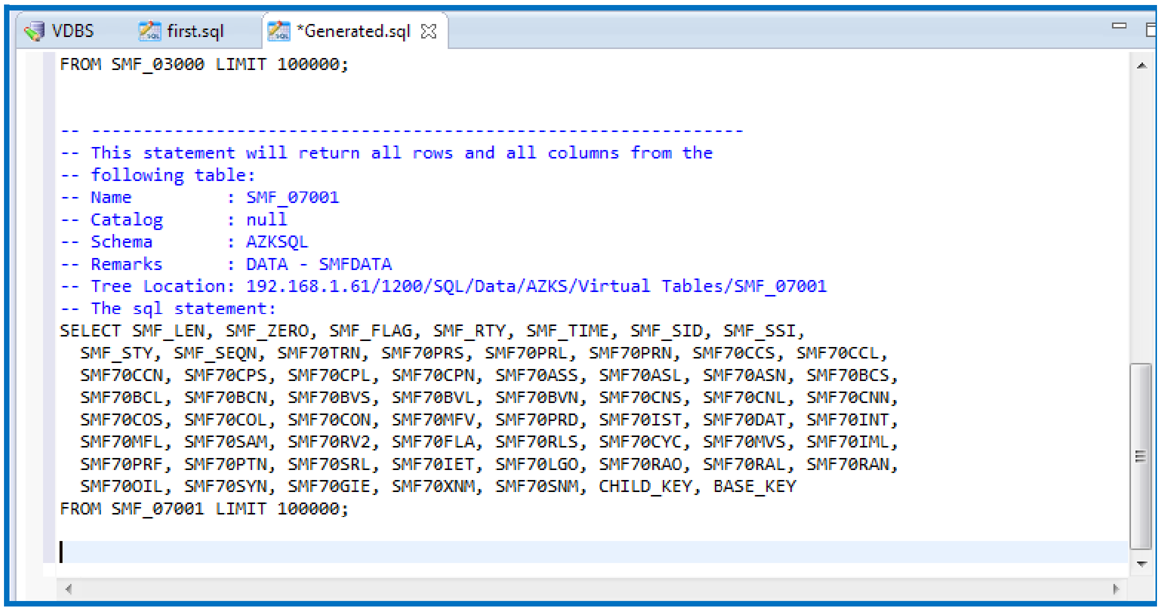


Host: 192.168.1.61 (Port 1200)
Server: AZKS
Version: 01.01.00.0000

SMF_07001
SMF_LEN [INTEGER]
SMF_ZERO [INTEGER]
SMF_FLAG [VARCHAR(8)]
SMF_RTY [SMALLINT]
SMF_TIME [TIMESTAMP]
SMF_SID [VARCHAR(4)]
SMF_SSI [VARCHAR(4)]
SMF_STY [SMALLINT]
SMF_SEQN [VARCHAR(10)]
SMF70TRN [INTEGER]
SMF70PRS [INTEGER]
SMF70PRL [INTEGER]
SMF70PRN [INTEGER]
SMF70CCS [INTEGER]
SMF70CCL [INTEGER]
SMF70CCN [INTEGER]
SMF70CPS [INTEGER]
SMF70CPL [INTEGER]
SMF70CPN [INTEGER]
SMF70ASS [INTEGER]
SMF70ASL [INTEGER]

As you can see in Figure 6, all the fields in the base part of a type 70 SMF record are listed. We will come back in a few minutes and discuss how Rocket have mapped the SMF records, but for now we want to concentrate on the mechanics of using the Data Studio, so bear with us for a few minutes (yes, yes, I know, you can't wait to try it out). If you right click on the table name, you will be presented with the panel shown on the right of Figure 6. The last option on that list is 'Generate Query'. If you click on that option, you will be presented with a panel asking if you want to execute the query now. Click Cancel.

Figure 7 - SQL Statements for Generated Query

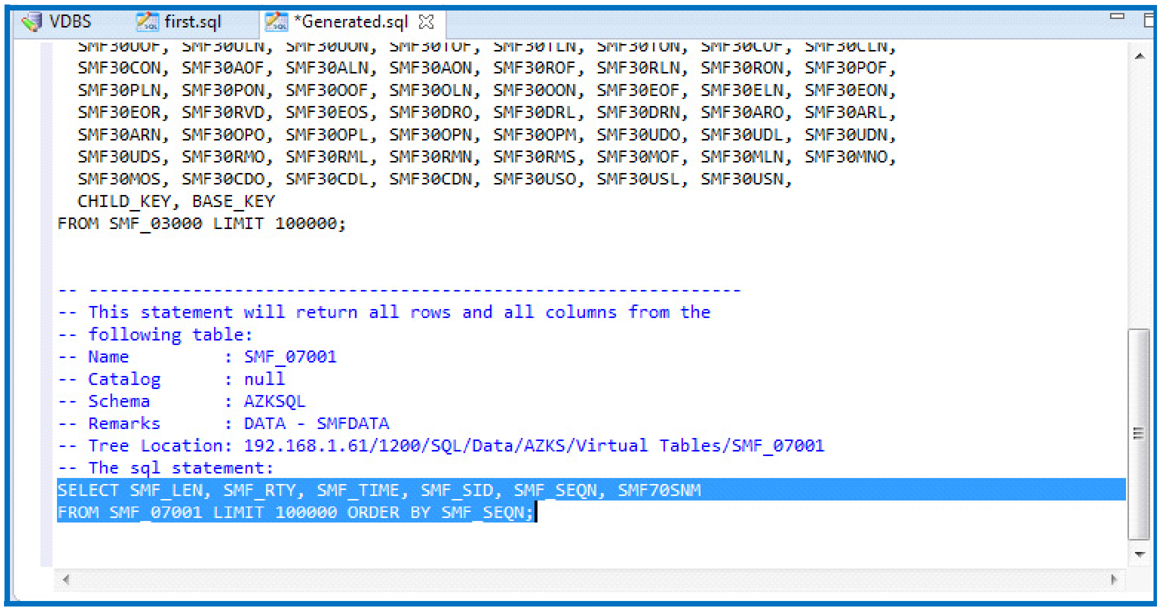

```
FROM SMF_03000 LIMIT 100000;


-- ---------------------------------------------------------------
-- This statement will return all rows and all columns from the
-- following table:
-- Name         : SMF_07001
-- Catalog      : null
-- Schema       : AZKSQL
-- Remarks      : DATA - SMFDATA
-- Tree Location: 192.168.1.61/1200/SQL/Data/AZKS/Virtual Tables/SMF_07001
-- The sql statement:
SELECT SMF_LEN, SMF_ZERO, SMF_FLAG, SMF_RTY, SMF_TIME, SMF_SID, SMF_SSI,
  SMF_STY, SMF_SEQN, SMF70TRN, SMF70PRS, SMF70PRL, SMF70PRN, SMF70CCS, SMF70CCL,
  SMF70CCN, SMF70CPS, SMF70CPL, SMF70CPN, SMF70ASS, SMF70ASL, SMF70ASN, SMF70BCS,
  SMF70BCL, SMF70BCN, SMF70BVS, SMF70BVL, SMF70BVN, SMF70CNS, SMF70CNL, SMF70CNN,
  SMF70COS, SMF70COL, SMF70CON, SMF70MFV, SMF70PRD, SMF70IST, SMF70DAT, SMF70INT,
  SMF70MFL, SMF70SAM, SMF70RV2, SMF70FLA, SMF70RLS, SMF70CYC, SMF70MVS, SMF70IML,
  SMF70PRF, SMF70PTN, SMF70SRL, SMF70IET, SMF70LGO, SMF70RAO, SMF70RAL, SMF70RAN,
  SMF70OIL, SMF70SYN, SMF70GIE, SMF70XNM, SMF70SNM, CHILD_KEY, BASE_KEY
FROM SMF_07001 LIMIT 100000;
```

After you click Cancel, the right part of the Data Studio window will turn blue. When you click in that pane, you will see that Data Studio has generated an SQL SELECT statement that will select the first 100,000 rows from that virtual table as shown in Figure 7. You can then remove the fields that you are not interested in, you can add WHERE clauses, and so on. When you believe that your query is ready to run, swipe over all the statements that you want executed and press F5.

In Figure 8, you can see that we trimmed down the number of fields (columns) that we want to retrieve, and we want to order the list by the sequence in which the type 70 records appeared in the input file (the SMF_SEQN field).[9]

Figure 8 - Tailored SQL Statements


```
  SMF30UOF, SMF30ULN, SMF30UON, SMF30TOF, SMF30TLN, SMF30TON, SMF30COF, SMF30CLN,
  SMF30CON, SMF30AOF, SMF30ALN, SMF30AON, SMF30ROF, SMF30RLN, SMF30RON, SMF30POF,
  SMF30PLN, SMF30PON, SMF30OOF, SMF30OLN, SMF30OON, SMF30EOF, SMF30ELN, SMF30EON,
  SMF30EOR, SMF30RVD, SMF30EOS, SMF30DRO, SMF30DRL, SMF30DRN, SMF30ARO, SMF30ARL,
  SMF30ARN, SMF30OPO, SMF30OPL, SMF30OPN, SMF30OPM, SMF30UDO, SMF30UDL, SMF30UDN,
  SMF30UDS, SMF30RMO, SMF30RML, SMF30RMN, SMF30RMS, SMF30MOF, SMF30MLN, SMF30MNO,
  SMF30MOS, SMF30CDO, SMF30CDL, SMF30CDN, SMF30USO, SMF30USL, SMF30USN,
  CHILD_KEY, BASE_KEY
FROM SMF_03000 LIMIT 100000;


-- ---------------------------------------------------------------
-- This statement will return all rows and all columns from the
-- following table:
-- Name         : SMF_07001
-- Catalog      : null
-- Schema       : AZKSQL
-- Remarks      : DATA - SMFDATA
-- Tree Location: 192.168.1.61/1200/SQL/Data/AZKS/Virtual Tables/SMF_07001
-- The sql statement:
SELECT SMF_LEN, SMF_RTY, SMF_TIME, SMF_SID, SMF_SEQN, SMF70SNM
FROM SMF_07001 LIMIT 100000 ORDER BY SMF_SEQN;
```

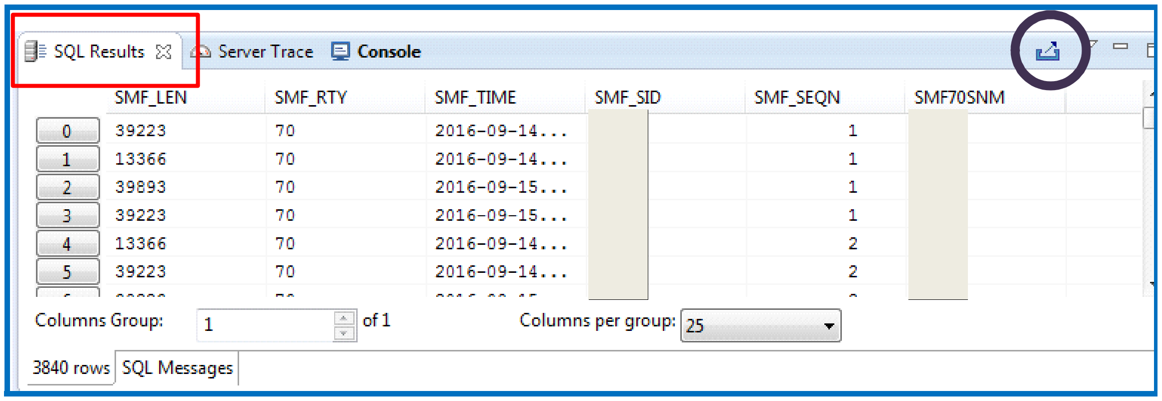
[9] At the time of writing, each reader task assigns sequence numbers independently, so with a default of 4 reader tasks, you would get 4 records with the same sequence number. Until this is resolved, the best way to get repeatable sequences of rows is to do an ORDER BY SMF_TIME.

NOTE: The input file in this case contains many SMF record types, but you will notice that our SELECT statement didn't do any filtering on record type. Because we are running the query against the virtual table for type 70 subtype 1 records, MDSS knows to ignore all other record types and subtypes.

When we press F5, we are presented with a prompt asking if we want to run the query in the background. In this example, the query ended before I had made up my mind about whether I wanted to run it in the background or not.

When the query completes, the results are shown in the SQL Results tab in the bottom right pane of the window, as shown in Figure 9. On the top right of the pane, highlighted by the purple circle, you can see what looks like a little box with an arrow coming out of it. If you click that icon, you will be presented with a screen asking where you would like the csv containing the results to be stored.

Figure 9 - Query Results



	SMF_LEN	SMF_RTY	SMF_TIME	SMF_SID	SMF_SEQN	SMF70SNM
0	39223	70	2016-09-14...		1	
1	13366	70	2016-09-14...		1	
2	39893	70	2016-09-15...		1	
3	39223	70	2016-09-15...		1	
4	13366	70	2016-09-14...		2	
5	39223	70	2016-09-14...		2	

Columns Group: 1 of 1   Columns per group: 25

3840 rows   SQL Messages

So, there you have it. With the investment of maybe half a day to get this product installed, minimally customized, and started, you can now run SQL queries against sequential SMF data sets. If you are fortunate enough to already have SAS and MXG, your reaction might be 'so what?'. We try to avoid talking about prices or finances in the Tuning Letter, but it is hard to ignore the fact that the only cost for Spark and MDSS is the Service and Support, and that is optional. For sites that do *not* have SAS and MXG, Spark and MDSS provide an ease of access to SMF data that is unheard of, especially at this price point.

The next section in this article will discuss how Rocket have implemented SMF support in MDSS, and provide some tips based on our experiences.

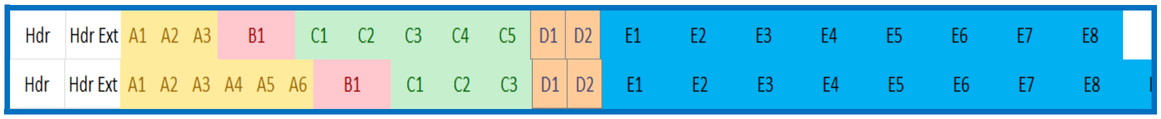
Accessing SMF Data with MDSS

The information in this section is based on my experiences with using the MDSS half of the Spark package to extract information from sequential data sets containing SMF records. It is not presented in any particular order; rather, it is just a list of things that I hope full be helpful when you start using Data Studio to access SMF data.

How SMF Records are Mapped in MDSS

While Data Studio makes it very easy to access fields in SMF records, you still need to understand the underlying data and how it is structured. SMF records are typically variable length. This allows the same record type to contain varying amounts of information. An example is shown in Figure 10. In this case, we have the same record type, but the first one is reporting on 3 ‘A’ items, while the second one is reporting on 6 ‘A’ items. The first one has 5 ‘C’ items, while the second one has 3 ‘C’ items. This idea of having a variable number of instances of certain parts of the record is called repeating sections. The Hdr Ext near the start of each record contains information about the length of each repeating section, its offset into the record, and the number of instances of that section. In this example, you can see that the ability to have repeating sections results in the first record having a total length that is a little shorter than the second record.

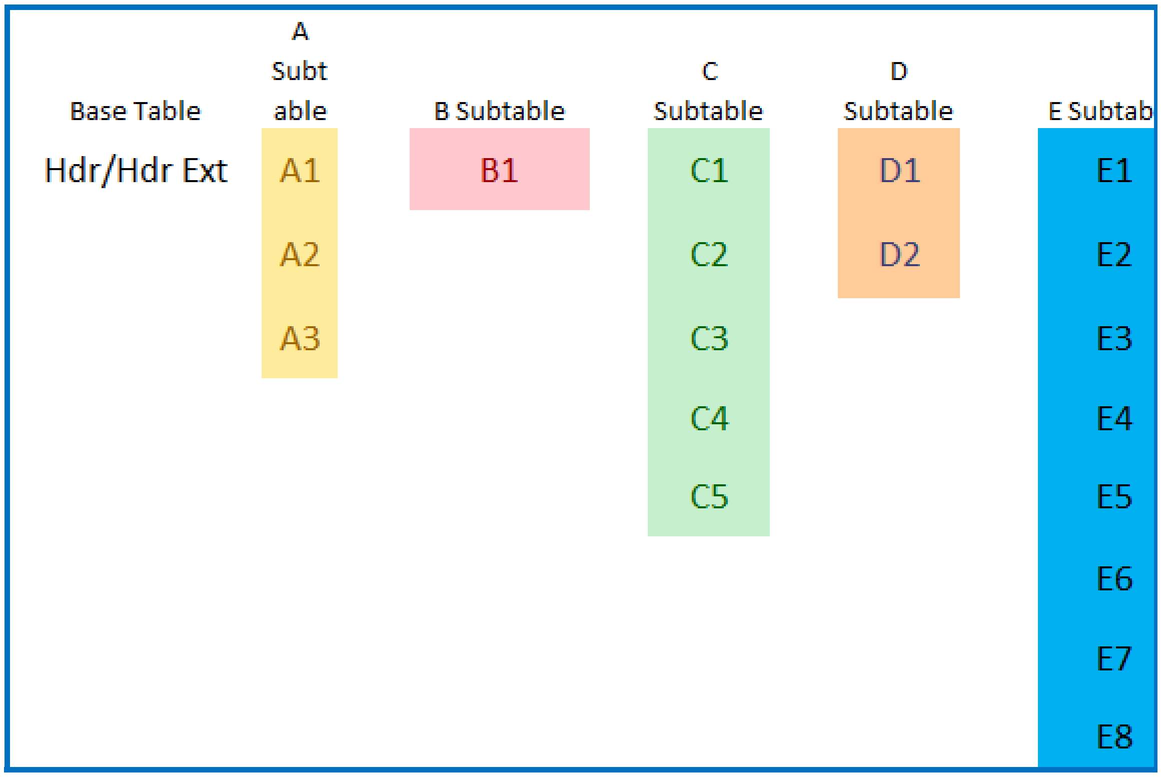
Figure 10 - Typical SMF Records



This mechanism is very flexible and a very effective way to present the maximum amount of information in the smallest amount of space. Unfortunately, SQL doesn’t want anything to do with variable length records. In the SQL world, everything fits in tables with a pre-defined number of columns. So, how do we merge these two worlds?

The normal solution when storing SMF records in DB2 is to create separate tables for each repeating section. This results in something like the structure shown in Figure 11 on page 17, with each column representing a separate table. Because each repeating section has a fixed number of fields, this fits nicely with the SQL requirement to have a fixed number of columns in each table.

Figure 11 - Mapping an SMF Record into Tables and Subtables



The challenge with this model is - how do you maintain the relationship between the rows in the various tables? When all the information was in a single record, the time, date, and owning system for every field was easy to determine because that information is in the header of each record. But when you move the repeating sections out into separate tables, that relationship is lost. The solution is to add a field to the table containing the base part of the record and a corresponding field in the tables containing the repeating sections.

If you look in Data Studio, you will see that nearly every record type that has been mapped has a number of virtual tables. The will be one called SMF_tttss, and one or more called SMF_tttss_aaaaaa, where ttt is the record type (in decimal), ss is the subtype, and aaaaaa is a string of characters and numbers that indicate the repeating section that resides in that table. For example, the type 70 subtype 1 record is loaded into the following tables:

- SMF_07001
- SMF_07001_SMF70AID
- SMF_07001_SMF70BCT
- SMF_07001_SMF70BPD
- SMF_07001_SMF70CIS
- SMF_07001_SMF70CPU
- SMF_07001_SMF70CTL

In MDSS terminology, the table that contains the record header is called the base table and the tables that contain the repeating sections are called subtables. The base table contains a generated column called 'CHILD_KEY', and the subtables contain a generated field called 'PARENT_KEY'. All the rows in the base table and the subtables that have the same CHILD_KEY and PARENT_KEY values came from the same SMF record.

If you want to extract fields from the base table (time, date, and system name, for example) and from repeating sections that were in the same record, you would do a JOIN using the PARENT_KEY and CHILD_KEY fields - for example:

```
SELECT SMF_TIME, SMF_SID, SMF_SEQN, SMF70VPA, SMF70BPS,
 FROM SMF_07001 A0
 JOIN SMF_07001_SMF70BPD A9
   ON A0.CHILD_KEY = A9.PARENT_KEY;
```

If you are not experienced with SQL (as I am not), you might be surprised that the above Select might return hundreds of rows for each type 70 record for each interval. SMF_07001_SMF70BPD contains the PR/SM Logical Processor data section, and that section has one instance for every logical CP for every LPAR on that CPC. So when you run the Select above, you will get x rows returned, with each row containing the SMF70VPA (logical processor number) and SMF70BPS (logical weight factor) fields from a different instance of the Logical Processor data section, along with the SMF_TIME, SMF_SID, SMF_SEQN fields from the SMF header section.

This example is a good illustration of the power of MDSS and also the importance of understanding the underlying data.

If you are familiar with SMF records, you might have noticed that the column names used for the SMF header fields are a little different to those used in the mapping macro and the SMF manual (for example, MDSS uses SMF_SID, but the manual uses SMF70SID). In normal processing, this should not be relevant. If you did a join on two virtual tables and wanted to extract the header fields, you would simply prefix the field name with the appropriate table name - for example, SMF_07001.SMF_TIME and SMF_07201.SMF_TIME.

Results Sequence

While getting familiar with Data Studio, I ran a Select against just two fields, and got the rows back in the sequence of SYS8, SYS6, SYS9, and SYS1. Then I added one more field, in the same table, and ran the Select again. This time, I got the rows back in a different sequence. The input data set was exactly the same, and the only change to my Select was to add a field - I didn't add any ORDER BY or WHERE clauses or any other changes.

Apparently this is standard SQL. If you run two slightly different queries against the same table, there is no guarantee that the rows will be returned in the same sequence. If the sequence is important to you, you should use an ORDER BY statement to specify the column that you want to sort the output on. If you want the rows in the same sequence that they exist in the input data set, specify ORDER BY SMF_SEQN (SMF_SEQN is a generated field that contains the sequence of that record in the input file).

SQL Tips for the Uninitiated

I know nearly nothing about SQL, but with the help of Google, I have been able to string together enough statements to be able to extract most of the information I need from various SMF records. Because at this stage in the implementation we are limited to using SQL statements, there is only so much that we can do. And there are probably things that could be achieved in an easier manner using a program than by using just SQL statements. Nevertheless, with a small amount of SQL 'expertise', the Data Studio does provide a powerful tool for ad-hoc queries against various SMF record types. If you do not have much experience with SQL, I would recommend investing $20 in a copy of SQL For Dummies, it will save you time and much head-scratching.

Just to get you started, a few SQL statements that you are likely to become very familiar with:

JOIN	To create one logical table from two virtual tables.
UNION	To let you concatenate two 'tables', which, in our case, means two data sets (GDG -1 and GDG -2, for example).
WHERE	Used to filter the rows that will be returned (for example, WHERE SMF_SID=MVSA).
ORDER BY	Control which fields the returned data is sorted on.

Which SMF Record Types are Supported?

Rocket is constantly developing support for additional SMF record types and shipping PTFs to deliver that support to customers. If you are trying to determine if support for a given record type is available on your system, the easiest way to do that is simply to go into the Data Studio and scroll through the list of supported virtual tables. That information is retrieved in realtime from MDSS, so it is an accurate reflection of the support that is installed on your system.

There are two ways to get a list of the supported record types - the Data Studio, and the ISPF interface

You can also get a list of the record types and the field names (but not the subtable names) using the ISPF interface. From the primary MDSS menu, select Option D (Data Mapping), then Option 1 (Map Display). There will be a row for each record type and subtype. Enter an X beside the base table you are interested in and you will be presented with a list of all the fields in that base table and all of its subtables. By scrolling to the right (PF11) you can also see the definition of each field (its format, length, offset, and so on). There is no FIND command, which would be nice given the number of lines in some of the displays. However, if you know the entire string that you are looking for, you could enter something like 'L SMF_TIME' to find the row containing SMF_TIME.

Access Control

MDSS uses your RACF ID from your TSO session or the RACF ID you used to sign on to Data Studio when accessing the data sets for your queries.

If you want to protect access to the MDSS dialog, you need to update the RACF Class Descriptor Table to add a new RACF CLASS called AZKS. The names of the resources within that class, the levels of access that are required, and the mechanism for enabling this checking in MDSS are described in the Security chapter of the *Spark Administration Guide*, SC27-8451.

How to Control the Level of Parallelism?

There are multiple places in MDSS where various levels of parallelism are supported. Because we are concentrating on MDSS in this article, we will leave the ones that are related to Spark until the next issue.

To maximize throughput, MDSS tries to use a number of parallel tasks to process the file that the query is being processed against. Let's say that it is a 4000-cylinder sequential data set residing on disk. The default number of read tasks in MDSS is the lesser of 4 or the number of zIIP engines. So MDSS will process the first 1000 cylinders with one task, the next 1000 with another task, the third 1000 with a third task, and the last 1000 with a fourth task. All these tasks run in parallel, thereby reducing the elapsed time to read the entire data set.

If there are two queries running in parallel, each query would have four tasks, and four dedicated sets of buffers that the requested records will be read into.

You can control the maximum number of tasks for each query using the ACIMAPREDUCETASKS parameter. I don't believe there is a way to control the maximum number of queries that can run in parallel.

Controlling MDSS Memory Usage

There are multiple parameters that influence the amount of memory that MDSS can use. One of my to-do's for the next Tuning Letter is to get a better understanding of all those parameters and the precise impact of each one.

However, to get you started, it might be helpful to know that, by default, MDSS uses a 4 MB buffer for each read task. And, by default, each query has 4 read tasks, so that is 16 MB of (above-the-bar) buffer space for each concurrent query.

Until we get more real world customer experiences, it might be best to accept the default values, and use MEMLIMIT to place an acceptable limit on the amount of memory that MDSS can allocate. Be aware, however, that if the volume of work being passed to MDSS is such that more than that amount of memory would be required, MDSS will abend with an out-of-memory message.

Use a MEMLIMIT to protect yourself from MDSS consuming too much memory

Selecting Different Input Data Sets

If you were accessing SMF data sets in a traditional production environment, you would probably have a set of JCL that references the -1 generation of your SMF offload data sets. In such an environment, the method of defining SMF data sets to MDSS that we described previously may be sufficient.

However, if you want to be able to easily use different data sets, which is most likely to happen while you are testing, that method is not really ideal. In that case, MDSS supports a mechanism whereby you can specify the data set name on the Select statement in Spark or MDSS. It is actually very simple, but it requires a change to the rules members.

On your Select statement, instead of coding FROM SMF_07001, you would code FROM SMF_07001__KYNEF_TEST_ALLDATA, where the data set you want to run the query against is KYNEF.TEST.ALLDATA. There are two important points here:

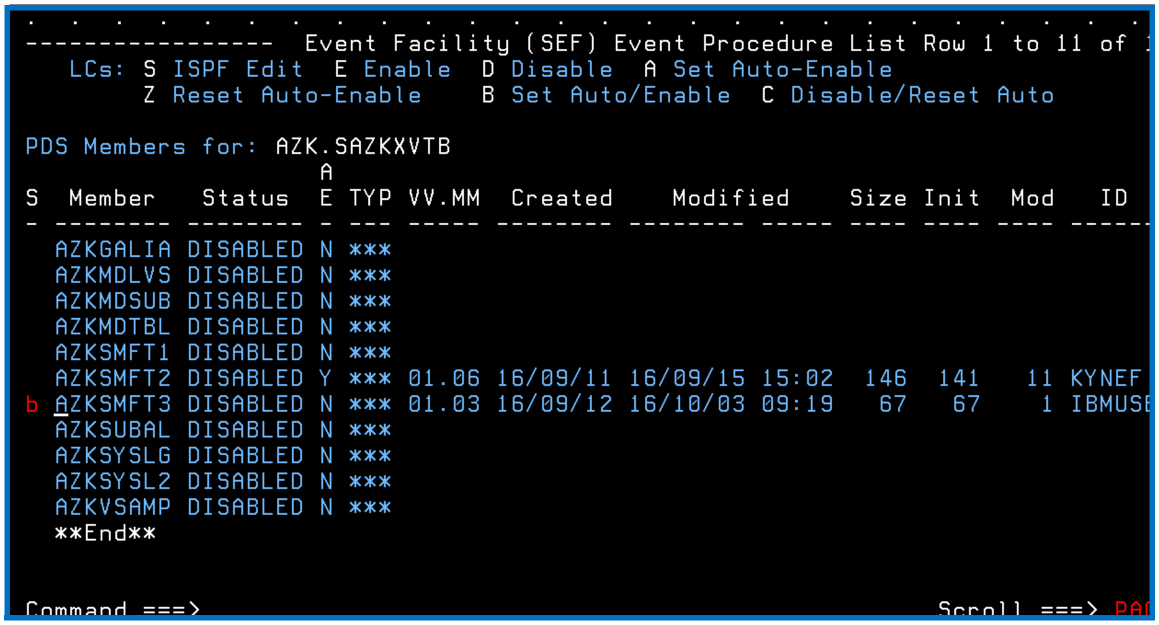
- You *must* have a double underscore between the table name and the data set name.
- In the data set name, you must use underscores between the qualifiers, rather than periods.

The other change you need to make is to go into the Virtual Tables rules in the MDSS ISPF interface and Enable the AZKSMFT3 rule.

TIP: If you change the value of any of the MDSS global variables, make sure to stop and restart the connection between Data Studio and MDSS. If you forget to do this, the new variable values will not be picked up.

Something else to consider is that, by default, the virtual table rules are DISABLED every time you stop and start MDSS. When you get to the point that you are happy with the rules, we recommend that you use the E.2 option in the ISPF interface to specify that the rule member(s) you want to use should be automatically enabled every time MDSS is started. You do this by entering a ‘B’ beside the rule name, as illustrated in Figure 12.

Figure 12 - Controlling Virtual Table Rules



```
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
-----------------  Event Facility (SEF) Event Procedure List Row 1 to 11 of 1
   LCs: S ISPF Edit  E Enable  D Disable  A Set Auto-Enable
        Z Reset Auto-Enable    B Set Auto/Enable  C Disable/Reset Auto

PDS Members for: AZK.SAZKXVTB
                    A
S  Member   Status  E TYP VV.MM  Created    Modified     Size Init  Mod   ID
- -------- -------- - --- ----- -------- -------- ----- ---- ---- ---- ------
  AZKGALIA DISABLED N ***
  AZKMDLVS DISABLED N ***
  AZKMDSUB DISABLED N ***
  AZKMDTBL DISABLED N ***
  AZKSMFT1 DISABLED N ***
  AZKSMFT2 DISABLED Y *** 01.06 16/09/11 16/09/15 15:02  146  141   11 KYNEF
b AZKSMFT3 DISABLED N *** 01.03 16/09/12 16/10/03 09:19   67   67    1 IBMUSE
  AZKSUBAL DISABLED N ***
  AZKSYSLG DISABLED N ***
  AZKSYSL2 DISABLED N ***
  AZKVSAMP DISABLED N ***
  **End**


Command ===>                                                  Scroll ===> PAG
```

Views

If you don’t fancy wrangling with SQL to reconstitute the SMF records that you know and love, the nice people at Rocket have kindly provided hundreds of views that will do the Joins for you. The views generally correspond to the subtables. So, for example, if you wanted to extract all the fields from the header section of the type 70 record, and the PR/SM Logical Processor data section (which is in subtable SMF_07001_SMF70BPD), you could use the view called SMF**V**_07001_SMF70BPD. The V in the fourth character of the tablename marks this as a view.

While it was not available at the time of writing, we believe that Rocket is working on the ability to specify a data set name when using views, just as you can when using virtual tables. The same mechanism is used - you use a double underscore to concatenate the DSN to the view name, replacing the periods in the data set name with single underscores.

Batch Queries

One of the questions I had was if it is possible to run multiple queries in a single iteration. In fact, not only is it possible, you have a choice of ways to achieve this.

One option is to code multiple sets of Select statements in Data Studio, swipe over all the statements that you want to execute, and press F5. All the queries will be run, and the results of the queries will be presented as multiple tabs in the SQL Results pane.

The other option is to use the AZKXMAPD batch program (the one that is used by the AZKIVVS1 job referenced previously). You could have multiple steps in the job, each executing AZKXMAPD and each writing its output to the data set referenced by the FMT DD card.

Viewing and Changing MDSS Parms

As mentioned previously, MDSS has over 1000 parameters. Fortunately, most of these can be ignored. However, if you would like to view the parameters, their meaning, and their current setting, the best way is to use the MDSS ISPF interface. From the primary panel, enter ‘C’ (AZK Admin). That will present you with a long list of further options. Select option 2, AZK Parms.

Because there are so many parameters in MDSS, they have arranged related parameters into groups. So when you select option 2, you will see a list of about 24 groups, with the group name and a short description of each group. If you are looking for a particular parameter and are not sure which group it is in, select the PRODALL group by placing a D beside the group name. On the subsequent panel, use F11 to move to the right, then enter ‘Sort name’. This will sort the list by parameter name.

Placing a D beside any parameter will present you with a description of that parameter. That screen also shows you the minimum and maximum values for each parameter and an indicator of whether the parameter can be dynamically updated or not. Pressing F10 will bring you back to a list of short descriptions of each parameter, and the current setting. Parameters that can be changed dynamically are shown in red. To change a parameter, simply overtype the value with the new value.

Adding Your Own Record Definitions

The IBM z/OS Platform for Apache Spark User's Guide provides information to help you add definitions for your own data sets to MDSS.

If there is an SMF record type that you want to process with Spark or MDSS and it is not currently among the set of supported record types, we recommend that you first talk to Rocket to see if they have plans to add that record type. The format of SMF records is not trivial, and Rocket have developed tools and techniques to help them map SMF records in a consistent manner. While you could certainly do the mapping yourself, it makes more sense for everyone to have that mapping done one time, than to have each customer reinvent the wheel, and create definitions that are unique to them.

How to Look Up MDSS Messages?

If you have been looking through the product manuals, you will have noticed that there is no message manual. However, the documentation for all the MDSS messages can be viewed using the ISPF interface.

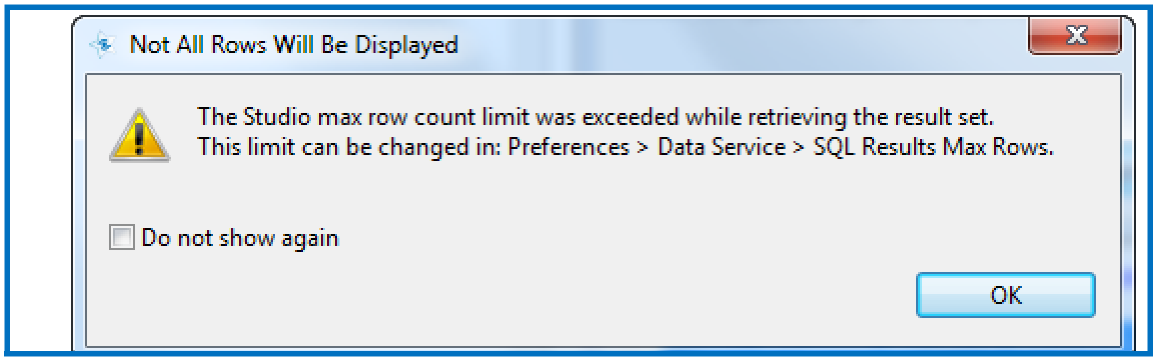
From the primary panel, enter ‘C’ (AZK Admin), then 2 (AZK Parms), then place a D beside the ‘PRODMESSAGES’ group and press Enter. You will be presented with a list of all the MDSS messages (1719 of them in our level of the code). Use the L (LOCATE) command to find the message you are interested in. When you have located your message, enter a D beside it and press Enter to see the message text, an explanation, and the suggested action.

Data Studio Limitations

It is important to remember that the intent of the Data Studio is to help you develop and test SQL queries that you will subsequently integrate into Spark queries. Many of the things that you will want to do with SMF data will require logic or data manipulation that is beyond the capabilities of pure SQL.

Also, by default, it returns relatively small amounts of data. Remember, the objective is to test your SQL statement. If the data set you are using for your test contains more data than the limit specified in the Data Studio, you will get a popup message like that shown in Figure 13 on page 24. For most testing, the default limits should provide an effective balance between returning enough rows to validate your query, while at the same time not consuming more resources than necessary.

Figure 13 - Data Studio Query Limits

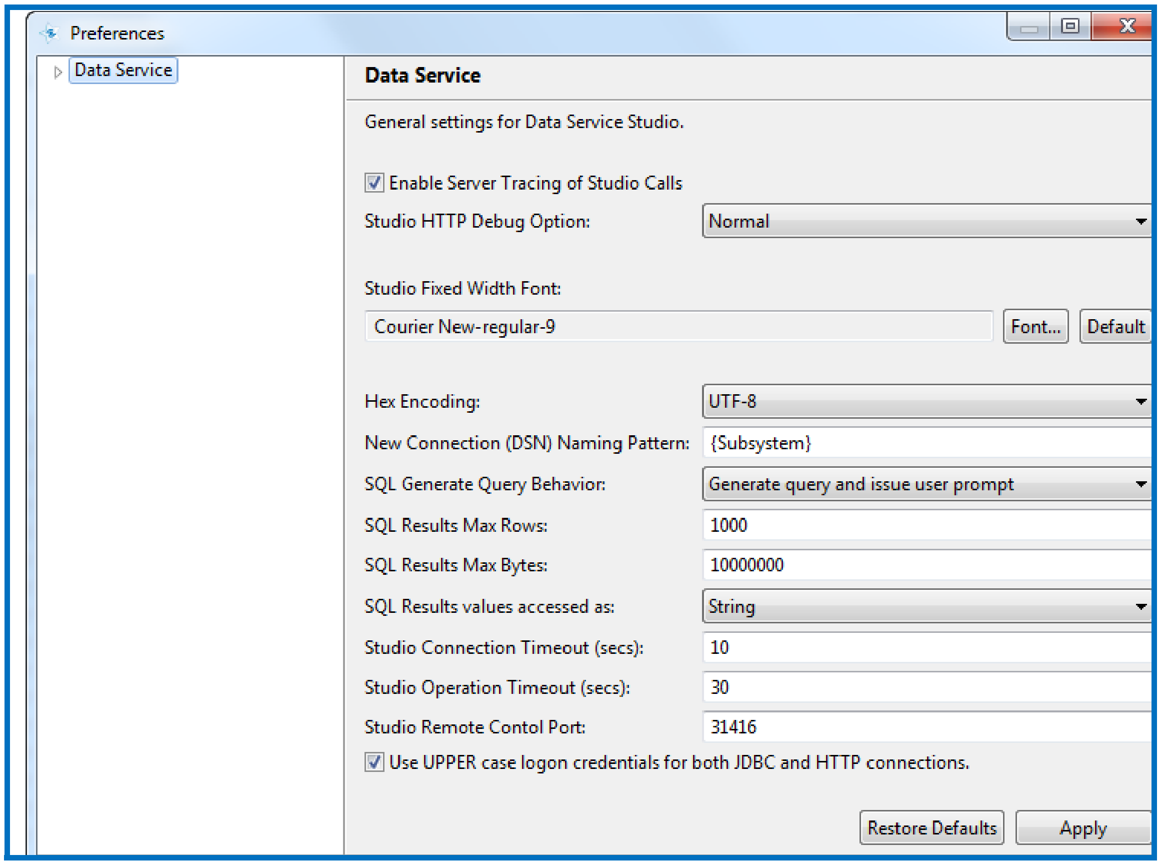


Not All Rows Will Be Displayed

The Studio max row count limit was exceeded while retrieving the result set.
This limit can be changed in: Preferences > Data Service > SQL Results Max Rows.

However, if you have a valid need to increase the limit, that is easy to do. (Ignore the text in the message, it appears to be out of date.) In the top right corner of the ‘SQL Results’ tab, there is a little down arrow. Click on that and select ‘Properties’. That will present the screen shown in Figure 14. Depending on the message in the popup window, you can increase the limit on the number of rows returned, or the number of bytes returned. Simply increase the value and click on ‘Apply’.

Figure 14 - Adjusting Data Studio Limits



Data Service

General settings for Data Service Studio.

☑ Enable Server Tracing of Studio Calls

Studio HTTP Debug Option:	Normal
Studio Fixed Width Font:	Courier New-regular-9
Hex Encoding:	UTF-8
New Connection (DSN) Naming Pattern:	{Subsystem}
SQL Generate Query Behavior:	Generate query and issue user prompt
SQL Results Max Rows:	1000
SQL Results Max Bytes:	10000000
SQL Results values accessed as:	String
Studio Connection Timeout (secs):	10
Studio Operation Timeout (secs):	30
Studio Remote Contol Port:	31416

☑ Use UPPER case logon credentials for both JDBC and HTTP connections.

If You Hit Problems

You know that something will not work, either because of a glitch in the code, or because of 'user error' (not you, of course, some *other* user..). When this happens, a very useful tool is the Server Trace function that you access via option B in the ISPF interface.

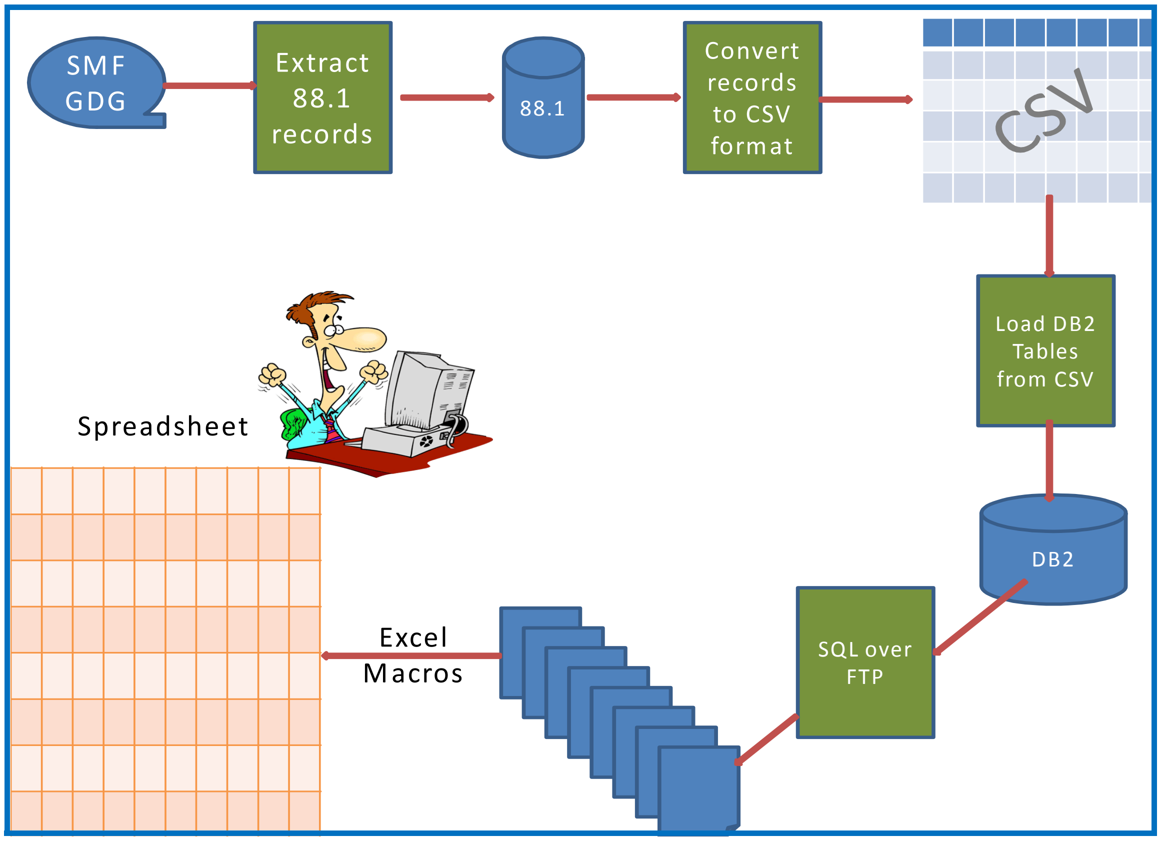
The initial display shows a 1-line summary for many of the MDSS actions (both successful and unsuccessful). The messages presented when you encounter a problem are not always the most user friendly. In those situations, go into the Server Trace, find the line that corresponds to your error (all lines are timestamped, so it is generally pretty easy to find the line that pertains to your problem). Sometimes the summary is sufficient to help you resolve the error. If it is not, move the cursor to that line and press Enter. This will result in multiple screens worth of diagnostic data. You might be able to resolve the problem using the information provided. If not, try searching the Rocket Software KnowledgeBase that is available to registered Rocket Software customers. If your problem still isn't resolved, open a PMR with IBM.

Sample Real World Exploitation

Many of our readers know our friend **Mario Bezzi** from IBM Italy. Mario is one of the smartest people I know, and definitely the most humble. Mario has a number of tools that he kindly distributes to customers. One of them is called SMF0881 - it processes the type

88 SMF records that are created by System Logger, resulting in a spreadsheet with many worksheets that provide insight into just about every perspective on Logger activity. The process is depicted in Figure 15 on page 26.

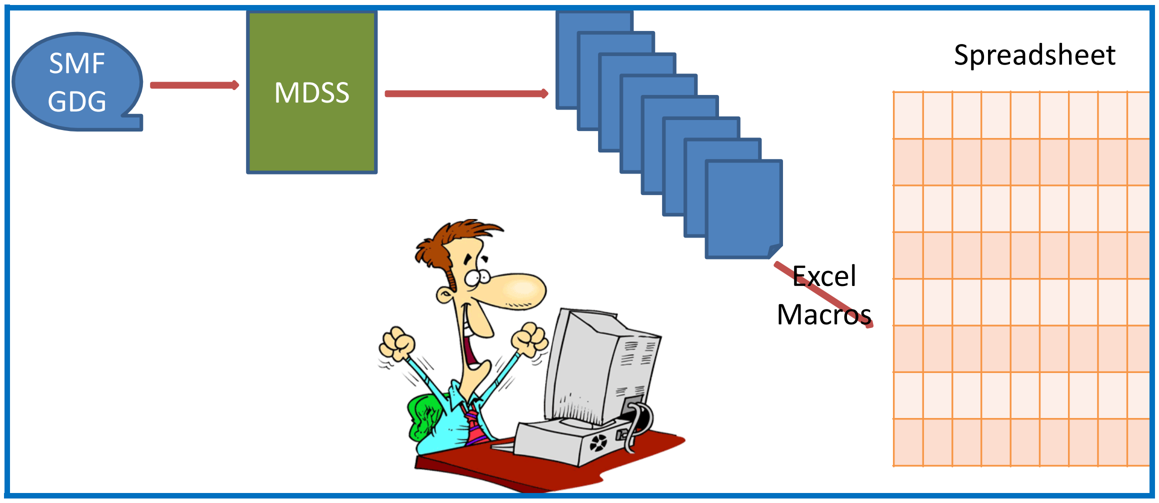
Figure 15 - SMF0881 Processing



You can see that it starts by reading a sequential data set containing SMF records, breaks them up into CSV format, loads them into multiple DB2 tables, and then runs a number of SQL queries over FTP, storing the results in a series of flat files on the user's PC. The last step is to run a set of Excel macros that read the flat files and populate a spreadsheet. The resulting spreadsheet contains invaluable information, and it is one of the things that I always do in a sysplex health check. However, you need to have DB2, and a friendly DBA that will give you access to create, load, query, and delete your own tables.

Now let's look at how the same end result could be achieved using Spark or MDSS.

Figure 16 - Creating SMF0881 Reports with MDSS



As you can see in Figure 16 on page 27, all the work involved in extracting the type 88.1 records, creating the CSV file, creating and populating the DB2 tables, and then running the SQL queries to create the flat files on the PC have been replaced with queries using MDSS. If you use Spark, the process would complete in a fraction of the time, and the creation of the flat files could be incorporated into a single Java program. If you were to do this using Data Studio, it would be a more manual process, but it still eliminates all the processing on z/OS that was taking place previously.

Mario has kindly agreed to Rocket reworking his queries to work with MDSS and we are hoping that they will soon be available as samples to illustrate the power of this solution. As an example, this is Mario's original query to extract the number of CF-only log streams that are reported on in the type 88 records:

```
select
  smf88lfl,
  (count(distinct(smf88lsn))) as CFOnly_lsn_c
  from SMFDATA.SMF0881_LGRDATA
  where int(left(smf88lfl,1)) = 0
   group by smf88lfl;
```

Because the MDSS implementation uses different table names and splits the fields up differently, some small changes (marked in blue) were needed to get it to work with MDSS:

```
select
   smf88lfl,
   (count(distinct(smf88lsn))) as CFOnly_lsn_c
   from SMF_08801 a0
   join SMF_08801_SMF88LSD a1
     on a0.CHILD_KEY = a1.PARENT_KEY
   where int(left(smf88lfl,1)) = 0
   group by smf88lfl;
```

You can see that the required changes were relatively trivial.

We hope to include this, and many other, sample queries on our website. If you have any home grown tools that load SMF records in relational tables and would be willing to share your queries with us, please send an email to technical@watsonwalker.com. None of us have time to reinvent the wheel, so the more people that are willing to share, the better is your chance of being able to find an existing query that does just what you need.

Considerations

Remember, this is the first release of the IBM z/OS Platform for Apache Spark. And despite the countless years that I have been working in the IT industry, I have yet to see any product that delivered perfection in its first release.

I also have to warn you that I'm an old z/OS guy that knows less than nothing about UNIX or networking, and nearly nothing about modern security. So, things that are daunting hurdles to me, might be a 'so what' to you. All I can do is give you my impressions. But I urge you investigate this product yourself and form your own opinion. Remember that there the only charge is for Support and Service, and that is optional.

So, having said that, what are the 'considerations'?

Documentation

The first one surrounds documentation. Most of the material in the IBM z/OS Platform for Apache Spark library is about the MDSS component. And the documentation that *is* available tends to be more in the line of reference manuals than 'how to' manuals. There are a lot of parameters for MDSS, the majority of which will probably never be changed by most customers. But there are still a lot of parms that affect performance and resource usage, and I believe that customers will need some advice to help them specify the best values for their environment. At the moment, that documentation doesn't appear to exist.

If you want information about the Spark component, there doesn't appear to be any centralized source of information about how to configure and tune Spark for a z/OS environment - all the documentation appears to be on the standard Spark website, which is naturally aimed at a distributed environment. As such, the terminology will be more familiar to someone with strong UNIX and network skills than to a traditional z/OS system programmer.

Also, from what I gather, some effort is required to optimize Spark performance for a z/OS environment - I don't believe that just accepting all the Spark default values will result in the best possible performance. Hopefully IBM will provide guidance to help customers optimize their Spark configuration.

Exploiters

As you have probably discerned by now, Cheryl and I are very enthusiastic about the Spark package. Even though we are still only scratching the surface, we see huge potential in this capability.

But the fact remains that the Spark package is an infrastructure, just like CICS. And just like CICS, it is only of value if you have applications that exploit it. It is still very new, so it is not reasonable to expect any vendors to have had a chance to explore it, let alone announce support for it.

But we believe that any product that processes SMF records today is a potential exploiter. In fact, we have already spoken to two such vendors, explaining why we believe it is something they should be investigating. However, any vendor will only develop products where there is a market. So, if you believe that the Spark package would provide a desirable platform for processing SMF data, we encourage you to talk to your vendors and let them know that this is something that you are interested in.

ISV SUPPORT: If you have vendor products that read SMF data, we recommend that you speak to them to let them know of your interest in Spark on z/OS.

While the team in Rocket is really doing an outstanding job of rolling out support for more and more SMF records, we would like a clear statement from IBM that they will not only continue to increase the range of SMF records that are supported, but also that they will commit to maintaining the mappings as SMF records are enhanced by new releases and APARs.

This has been a challenge to just about every product that processes SMF records, even those from IBM. Part of the problem is that each development group is responsible for maintaining their own SMF record definitions, so there is no centralized process that could be used to flag every change to every SMF record. However, if IBM could find some way to crack that nut, we believe that would give the Spark package a real competitive advantage, and would also make like a lot easier for its users.

Running in a Shared Environment

Anyone that is familiar with CICS or DB2, or z/OS in general, will very quickly recognize that Spark comes from a very different land. Things that we take for granted in a highly structured, intensively shared, environment like z/OS are irrelevant when you operate in a single-application, dedicated hardware, environment.

As a result, the levels of control that we expect for major subsystems are not necessarily in place yet (but see 'Futures' on page 30 for more information about planned enhancements).

As an example, the default number of engines that a Spark cluster will try to use is n-1, where n is the total number of zIIPs in the LPAR. That seems to be a little greedy, but I guess that is how it works. What about if I start a second cluster? Yep, he will also decide to help himself to n-1 again. And a third cluster? He'll also opt for n-1.

Naturally, as the system programmer, you would control the parameters that determine how the clusters will be configured and how much resource each can use. So, for example, you might set up one cluster that will be accessible to people analyzing SMF data, and another cluster that is accessible to people analyzing customer buying habits. However, if a user has access to submit a job to Spark, then they also have authority to set up their *own* Spark cluster (that is called a *local cluster*). Naturally, because it is your cluster, you can specify the zIIP and memory limits yourself. And, because it is *your* cluster, you obviously realize that it is *the* most important work running in that z/OS system, so you will define the settings accordingly. It is a little like giving everyone authority to a) define their own CICS region, and b) set up the WLM rules relating to that region. This might sound a little 'Wild West'ish' to you. But remember, the overwhelming majority of Spark instances are running on hardware where Spark is the *only* thing running on that hardware.

Another 'oddity' is memory management. When setting up the Spark cluster, you specify how much memory can be used by each executor, and by the cluster overall. Those limits determine how much memory can be used for the current subset of data that each executor is working on. The RDD, the cache that holds the superset of data that is being analyzed, resides in the Spark 'job' (they call it a driver). The only way to limit the memory use of the driver (as far as I can make out) is to specify a MEMLIMIT in the RACF profile of the user that submitted that job. Now, what happens if the query results in a set of data that is larger than the RDD? Well, obviously, it abends. Hmmm. I must dust off my VERY old copy of Introduction to MVS that discussed he concept of *virtual storage*. On the other hand, if my MEMLIMIT is larger than the amount of real memory in my LPAR, he will happily load data up to my MEMLIMIT, potentially causing the system to page like crazy.

So, the things that you or I might view as a shortcoming (the lack of central control) are really only issues when Spark is running in an environment with many other workloads. Given that this is the first release of Spark that runs on z/OS, it is understandable that it doesn't have features that are particularly suited to a mainframe environment. The good news is that IBM and Rocket are already looking to the future and enhancements that address some of these concerns are in the works.

Futures

The enhancements listed here are things that IBM and Rocket would like to add to the Spark offering. No timescale has been given, and naturally, there is no guarantee that these will be delivered. But both IBM and Rocket are taking this initiative very seriously, so we are hopeful that these enhancements will be delivered sooner rather than later.

Controlling Where Programs Execute

Regular readers might recall that, in Kathy Walsh's article about zIIP capacity planning in *Tuning Letter 2014 No. 4*, we discussed how different parts of IBM are providing conflicting guidance for how to specify the IIPHONORPRIORITY parameter. This parameter determines whether zIIP-eligible work can only run on zIIPs, or if it is allowed to overflow to general purpose engines if the zIIPs are overloaded. DB2 recommend setting it to YES (meaning that work can overflow to the GCPs), while the z/OS people recommend setting it to NO. Both have a valid reason for their recommendation, but that doesn't really help the customer that is caught in the middle.

I was concerned that, if you specify IIPHONORPRIORITY=YES in line with the DB2 recommendation, that you could potentially end up with your Spark work overflowing to the GCPs and eating a LOT of capacity on those engines. Not only would this consume valuable GCP capacity, it could also have the impact of increasing your peak rolling 4-hour average, and thereby increasing your software bill.

To protect customers from this situation, I believe that IBM is investigating a mechanism that will protect your GCPs (and your software bill) from the damage that a flood of Spark requests could inflict. If that solution provides a way to force Spark work to only run on zIIPs, that work will have to queue if it requires more capacity than is currently available in the zIIPs. However, I think that this would be the preferred option for most people.

Prerequisites

Officially, the Spark package requires a zEC12/zBC12 or later. However, we believe that IBM will support it running on a z114/z196 as well. Parts of the code has been optimized for zEC12 or z13. When the code runs, it determines if it is on a z13/z13s, a zEC12/zBC12, or a z196/z114 and automatically selects the appropriate path for that CPC. For example, if you are running on a z13, it can exploit the SIMD capability that is only available on z13 or z13s or later. If it is running on a zEC12, it selects a path that does not use SIMD.

While both Spark and MDSS *will* run on a general purpose CP, you obviously will want to run them all on zIIPs. If you have the IIPHONORPRIORITY parameter set to YES (as recommended by DB2 V11), then its is possible that the Spark or MDSS work could overflow to your general purpose CPs, potentially impacting your software bill. At the moment, the only way to eliminate this risk is to set IIPHONORPRIORITY to NO (which DB2 V11 doesn't like) or to assign a discretionary WLM Service Class to both Spark and MDSS (zIIP-eligible work in a discretionary service class will not be selected by a general purpose CP). However, there are rumors that enhancements are in the works to provide more flexibility in this area, so watch this space.

On the software side, while you *can* download the Spark code from the link provided on the developerWorks site, we strongly recommend that you order the IBM z/OS Platform for Apache Spark product (5655-AAB) from Shopz. 5655-AAB is more up to date than the version you would get from DeveloperWorks, it is SMP/E-supported, and, most importantly, it includes MDSS.

We strongly recommend ordering the Spark package from Shopz, rather than using the free download from DeveloperWorks

5655-AAB is a no-charge product. If you want Service and Support, you would order 5655-AAC. The Service & Support fee is *per CPC*. You can run Spark in as many LPARs as you like on that CPC, and it doesn't matter if the CPC is a zEC12 701, or a z13 7E1 - the price is the same regardless of the size of the CPC or how many LPARs you run it in.

The Spark package requires:

◆ z/OS 2.1 or later.

◆ IBM 64-bit SDK for z/OS, Java Technology Edition, Version 8 Refresh 2 Fix Pack 10.

- Bourne Again Shell (bash) version 4.2.53, or later.

  This can be downloaded from http://www.rocketsoftware.com/ported-tools/bash-4254.

At the time of writing, there is an OPEN APAR, PI70069, that will deliver a significant number of enhancements to MDSS. Unfortunately, due to the deadline for this Tuning Letter, we were unable to have an opportunity to try those enhancements. But if you are installing MDSS, make sure that you pick up the PTF for this APAR if it is available.

Make sure to get the PTF for 'mega-APAR' PI70069

References

We have only touched on a small part of the capabilities of the Spark package on z/OS. Future articles will cover the Spark half of the package and potentially move on to discuss how you might use Spark to perform realtime analytics with SMF data. In the interim, the following sources of information may be valuable to you:

- The Spark on z/OS Announcement letter is available at: http://www.ibm.com/common/ssi/cgi-bin/ssialias?subtype=ca&infotype=an&supplier=897&letternum=ENUS216-067#epubx.

- IBM White Paper about the use of Spark for running analytics on z/OS titled *Sparking an analytics revolution.*

- IBM Redbook SG24-8325, *Apache Spark implementation on IBM z/OS.*

The following are the Spark on z/OS product manuals:

- *Program Directory for IBM z/OS Platform for Apache Spark,* GI13-4318.

- *IBM z/OS Platform for Apache Spark Installation and Customization Guide*, SC27-8449.
  - There is also a White Paper that contains additional information to assist with the installation - it should be used in conjunction with the Installation and Customization Guide. The White Paper title is *Installing IBM z/OS Platform for Apache Spark.*

- *IBM z/OS Platform for Apache Spark User's Guide*, SC27-8450.

- *IBM z/OS Platform for Apache Spark Administrator's Guide*, SC27-8451.

- *IBM z/OS Platform for Apache Spark Solutions Guide*, SC27-8452.

- Doc APAR PI66047, *Documentation Updates for z/OS Platform for Apache Spark*.

- IBM White Paper 102609, *Installing IBM z/OS Platform for Apache Spark*.

Summary

Despite the years of downsizing and rightsizing, the fact is that there is still an enormous number of production applications that reside on z/OS. And not just any old production application, but the production applications that bring revenue into your company - the ones that pay your salary and provide the funding for your company to invest for its future. Today, much of that data gets sucked out of your mainframe for use by applications that previously didn't exist on z/OS or that were cost-prohibitive on z/OS.

From speaking to customers, it seems to be common for tens of copies of operational data to exist on distributed platforms. Even ignoring the cost of all the hardware and software to support all those instances, the risk of someone hacking into that data *must* be higher when you have multiple copies of it. Twenty years ago, before hacking and cybercrime were in the news every day, this might not have been a concern. But, when you listen to the news and read reports by security experts, you increasingly have the impression that it is not a question of *if* you will be hacked, it is simply a question of *when*. Or even worse, has it *already* happened and we just don't know it yet.

Against that background, having the ability to gain the insight your company needs to operate effectively, without the need to spawn off multiple copies of your mainframe data *must* be appealing. Being able to do so in a manner that offers superior performance to distributed solutions AND that is extremely cost effective is just the icing on the cake.

In this issue we have only scratched the surface of the Spark package. At the conceptual level, we think that Spark could have a significant impact on the future of z/OS. It provides a mix of technology, required skills, and cost of ownership that is difficult to dismiss. Rumor has it that IBM has a long (really long) list of z/OS customers that are interested in doing Proof of Concept with Spark, so it isn't only us that see the potential power of Spark.

Speaking more from our own perspective, the most interesting aspect of the Spark package is its support for SMF data. The packaging of definitions of most of the more interesting SMF records with MDSS gives us the ability to perform ad-hoc extracts of just about any data we want from nearly any SMF record. Previously we relied on a hodgepodge of products, tools, and even IDCAMS PRINT (in times of desperation).

Will Spark replace the array of existing products that process SMF data? Certainly not overnight. But, given Spark's price point, if I were a vendor that processes SMF data, I certainly would be looking to see how to exploit the power of Spark and MDSS in my product.

But apart from existing products and functions, we also need to learn to think about SMF data in a new way. It is no longer just a record of what happened in the past. When you combine the analytics capabilities of Spark, the access to SMF data provided by MDSS, and the new SMF streaming function in z/OS, SMF data potentially allows you to take realtime actions that were inconceivable until recently. An obvious application is in detecting hacking attempts in real time, blocking them before they penetrate your system, rather than trying to clean up the mess afterwards. IBM's operations analytics tools have already started using SMF data and with the recent announcement of the Common Data Provider for z/OS (which we will cover in the next issue), real time access to SMF data will play an ever-larger role in effectively managing your systems. Combining off-the-shelf products like Capacity Management Analyzer and IBM Operational Analytics for z with the power of Spark to create bespoke analytics designed specifically for your company promises a whole new life for SMF data.

I'm presenting a session at CMG this week titled '*Spark and SMF - Fasten Your Seatbelt, This is Going To Be a Wild Ride*'. The more I think about it, the more prescient I think that title will be.

We hope that you will join us for the next article in this series, where we move on to the next step in our deployment - setting up a Spark cluster on z/OS and combining our new-found SQL ‘expertise’ with our yet-to-be-discovered Java skills to get even more value from the access to SMF data that the Spark package provides.

Back when I was a child living in the US, about 100 years ago, I remember that one of my father’s favorite television ads was some man saying ‘Try it. You’ll like it.’. I don’t remember what it was that he was proposing that you ‘try’, but I would repeat his exhortation to try the Spark package for yourself. You never know, you might like it. You might even start a revolution in your company, with people analyzing data where it is created - now there’s a mind blowing idea for you.