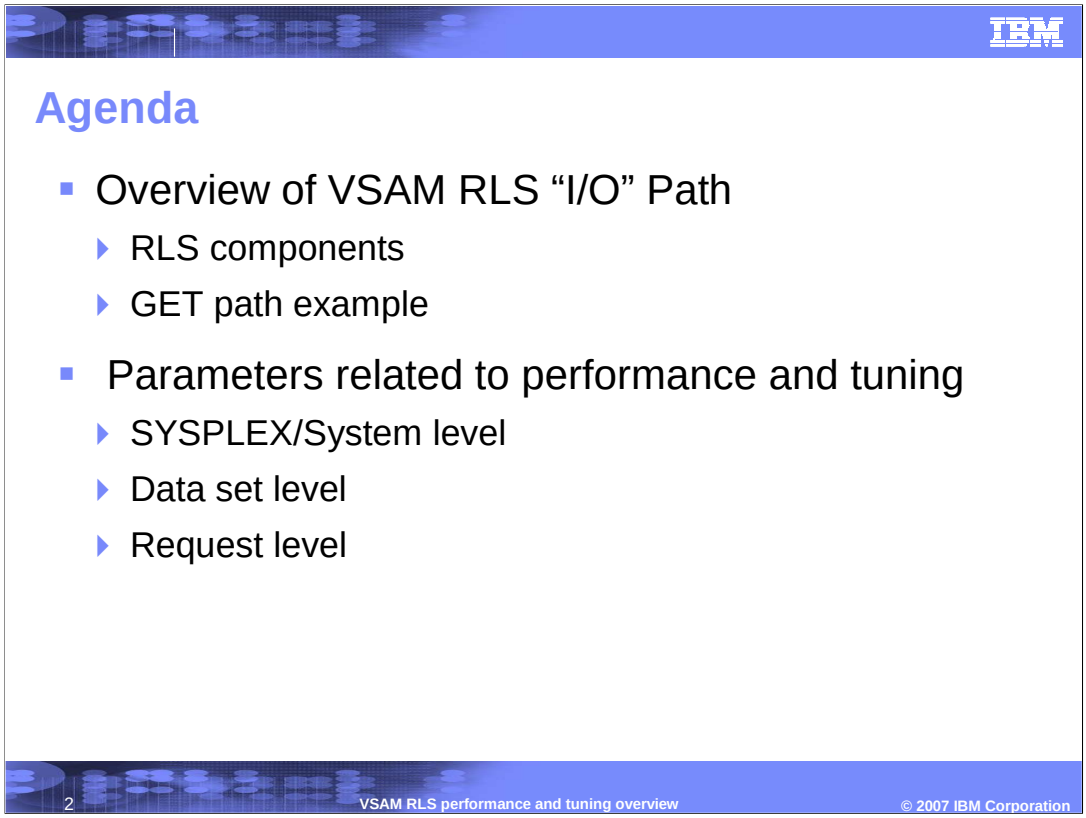
This is an overview of VSAM RLS I/O path and the parameters related to performance and tuning.



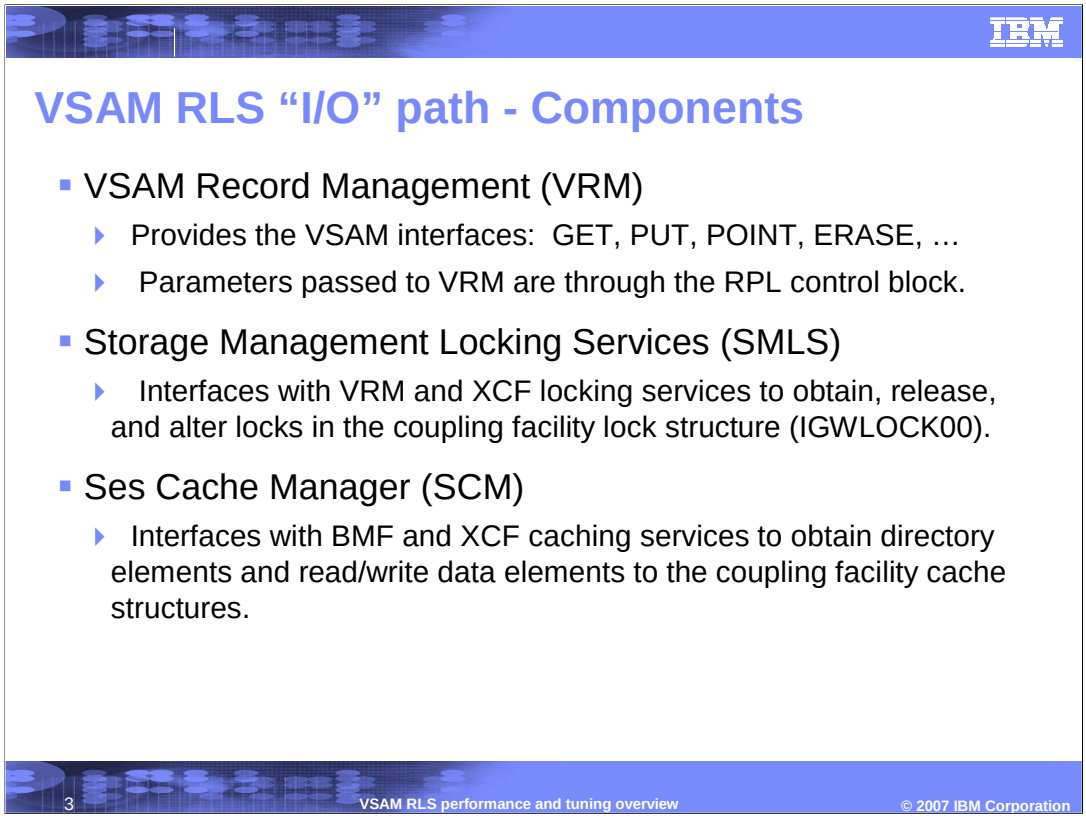
The subject of module is VSAM RLS performance and tuning. The most important thing is to look at what is called the performance path in RLS - also referred to as VSAM RLS I/O path. This is where you need to focus for performance and tuning.

In order to talk about the VSAM RLS I/O path, you need to understand the RLS component that makes that path. You need to be familiar with some terminologies that will be used. And then you will see a specific example of the I/O path. This presentation will focus on the GET path, because the GET path clearly shows you where you can get your benefit on performance.

In the WRITE path you may be doing some GET as well. You will learn how the GET path can be applied also to the WRITE path.

Once you have been introduced to I/O path, then you will be presented the parameters that are related to performance tuning. There are various parameters at different levels. There are sysplex or system parameters that you can set. There are dataset level parameters and other parameters that are even at the request level that you can set that help control the performance and tuning.

Once you are familiar with the parameters, continue to the next IBM Education Assistant module to see the specific examples of how you would set those parameters. There are also some structures that exist in the coupling facility for RLS. You will need to know how to size those structures correctly. You will learn about sizing a local buffer pool, the cache structure, and the lock structure and how they relate to performance and tuning.

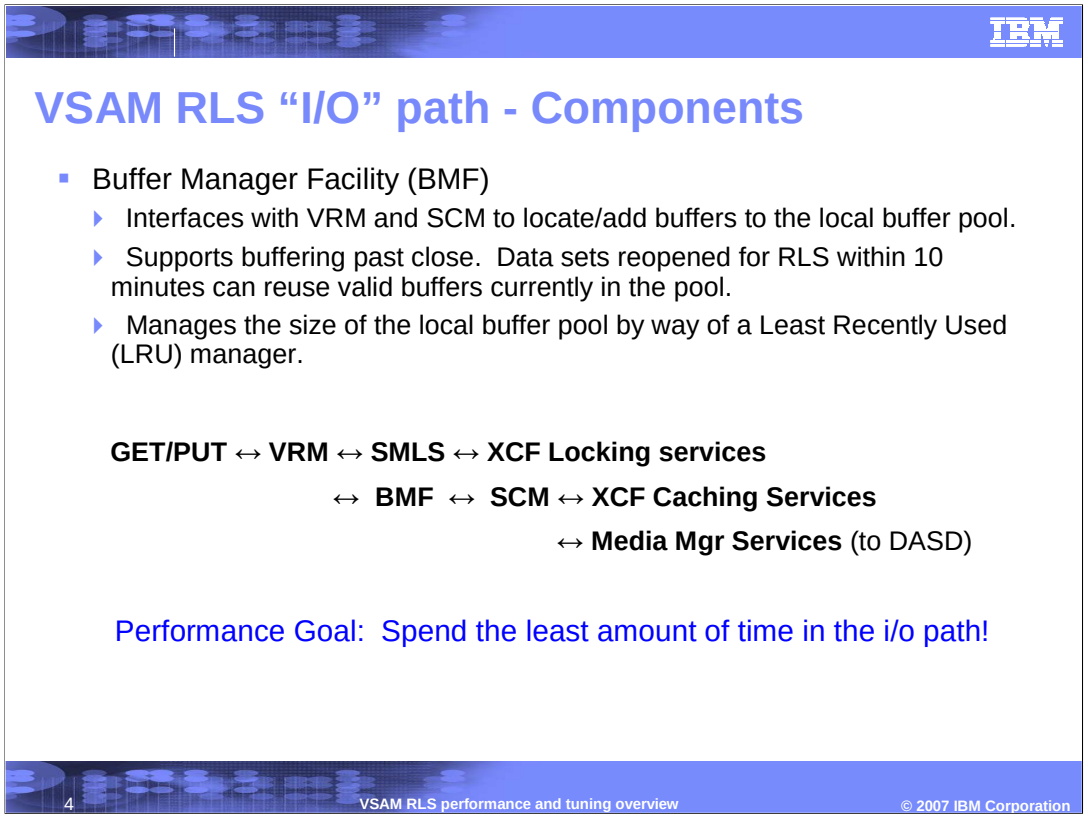


There are four main components that make up the RLS I/O path. The first one is called VSAM record management or VRM. So what this VSAM component does is this is a component that actually provide the interfaces to the VSAM application. So if you are writing a VSAM application, you would code these interfaces like GET, PUT, POINT, EARSE. And this is how you can communicate with RLS what type of request you want. When you code one of these macro interfaces, you are going to pass the parameters to VRM and that set of parameters is passing an RPL control block.

The next important sub-component that comes into play is Storage Management Locking Services or SMLS. This is the component that interfaces with VRM above and with XCF, which is an MVS component which provide locking services in the coupling facility. So with SMLS, XCF is called to obtain lock, release lock, alter lock. Again, the lock structure in the couple facility is called IGWLOCK000.

The next sub-component is Ses Cache Manager or SCM. This is a component that interfaces with BMF, which is the next component on the next page here.

XCF is another MVS component which manages the cache structure. SCM is going to call XCF caching services to obtain directory elements, or to read or write to the cache structure in the coupling facility.

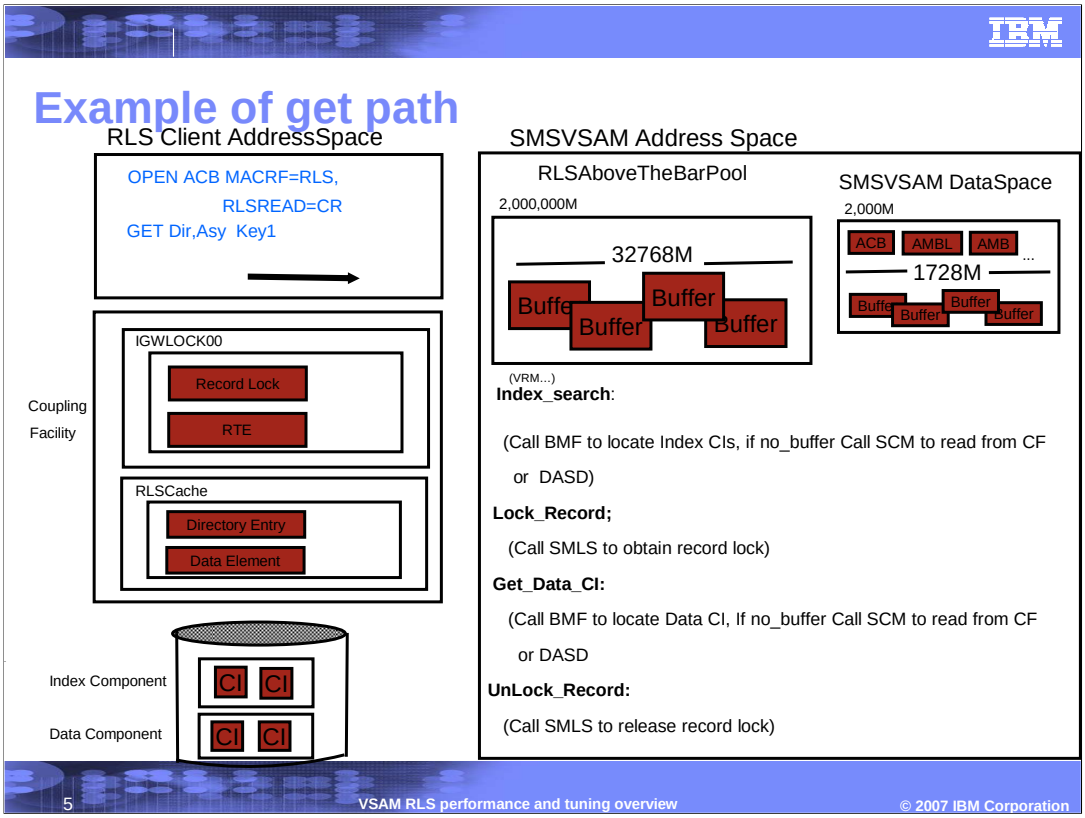


The last component, which is one of the most important one, is called the Buffer Manager Facility or BMF. BMF interfaces between VRM and SCM to locate the buffers and to move those buffers into the local buffer pool if necessary. One thing that is interesting about RLS with its buffering is the buffering path close. What that means is that when BMF reads buffers into buffer pool, those buffers will be kept in the buffer pool even though if you closed the dataset. So this is a very powerful buffering function. For example, when you open a dataset, you could read in a million of buffers into the buffer pool. You could close the dataset and if you reopen it within 10 minutes, you could still use those buffers that you have read in previously. All still will be valid.

BMF is responsible for reading these buffers and managing the space, including keeping track of the size of the buffer pool. BMF performs its functions with the Least Recently Used Manager or the LRU manager, which is a component within BMF.

So those are the four subcomponents that make up the I/O path. On the bottom of this page is a little diagram showing the relationships among the subcomponents. On the left you have done your VSAM request GET or PUT. So the first component VRM gets control. To perform record locking request, VRM calls SMLS, which will then call XCF to obtain a lock for this particular record. After that, BMF is called to find some buffers for this request. If the record cannot be found in the buffer pool, SCM will be invoked to see if it's in the cache. If it is not in the cache, the record will be retrieved from the DASD. Media Manger is placed down there, which is another component that DFSMS does the actual I/O.

The performance goal is to spend the least amount of time in the I/O path. If you look at that picture, you want to make the shortest path possible through that little diagram. The last step of going to DASD is the way extremely expensive than any other step. Skipping the last step will make a significant improvement in your time spending in the I/O path.



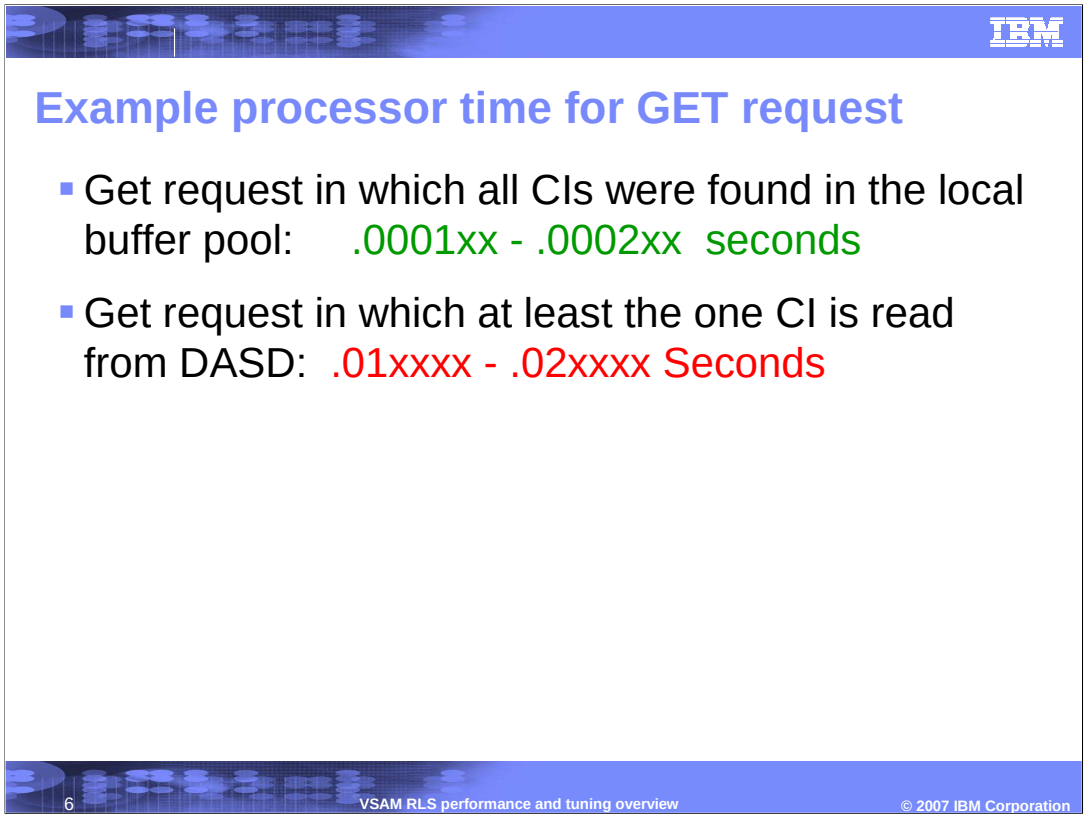
Here is an example of the Get path. This is the diagram previously looked but with greater details. This diagram shows one system in your sysplex. On the left, there is RLS client space, where the VSAM application is running. On the right, there is the SMSVSAM address space that has been initialized. At the top, there are two buffer pools which were created to hold the buffers. The first pool is the RLSAboveTheBarPool, which is a 64bit pool. The 64bit addressing will get you up to 2 terabytes of space. On the right, there is the SMSVSAM data space, which is used as a 31bit pool. So you can get up to 2 gigabyte worth of buffers in the 31bit pool. There are lines drawing through both buffer pools. In the RLSAboveTheBarPool, there is a line drawing at 32 gigabytes. The reason for that is while theoretically RLS can support up to 2 terabytes worth of data, RLS realistically has not gone passed 32 gigabytes yet. The threshold line is used to indicate the current supported maximum size. In the data space, the threshold line is set at 1.7 gigabytes. Space is reserved at the top above the line for control blocks.

This diagram also shows how VSAM RLS is performing buffering.

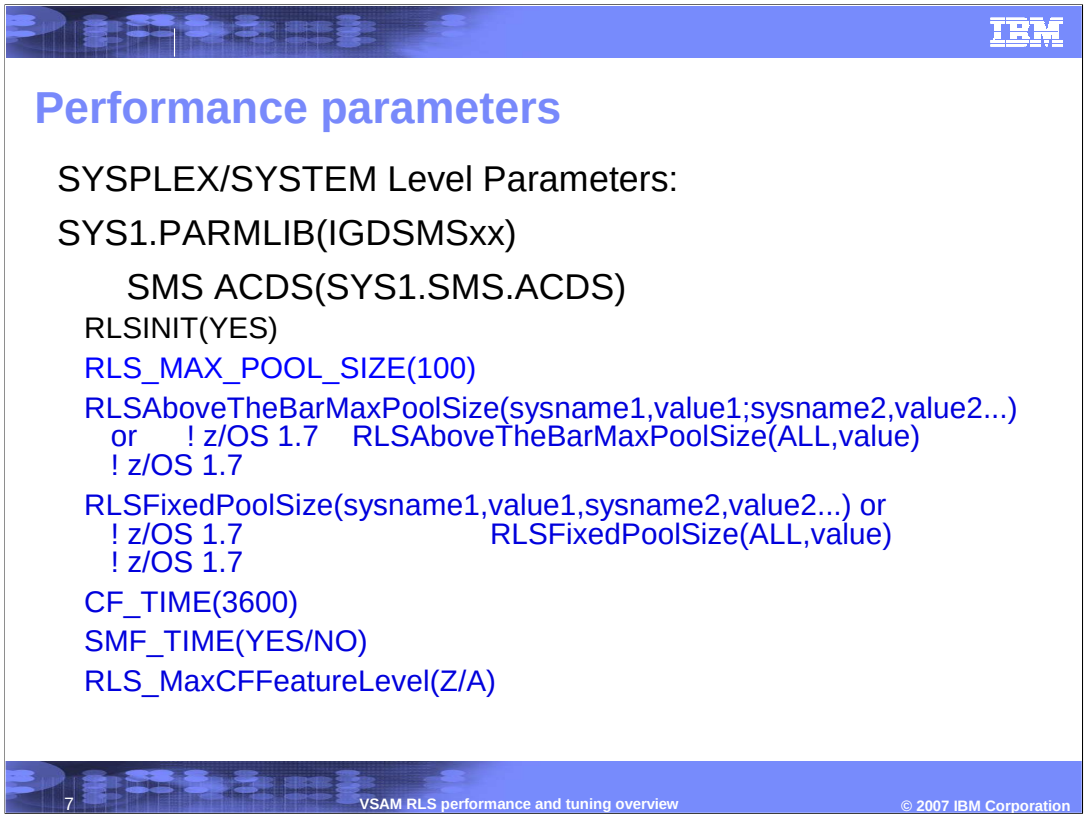
On the middle left, there is the coupling facility. RLS has two structures: the lock structure and the cache structure. Accessing the data in a dataset requires communication with these structures. Down on the bottom is the actual dataset. In this example, it's going to be a KSDS dataset. There are a data component and an index component. So, there are buffers and CI down there on the DASD. The goal is to get them into the buffer pool and not spending time going out to the DASD and locating those same buffers.

This example shows a Get request. To the left of the diagram is the client space. The client is going to open a dataset. An ACB is opened to do that. In this case, the ACB is asking for RLS access with RLSREAD field equals CR, which indicates read integrity request for the records to be read. That means when these records are read, these records cannot be updated by any other program at the same time. In order to perform read with integrity, serialization must be done on these records to prevent any one else from updating it while the records are being read. As a result, record locks will be obtained by SMLS component. Having not asking for reading integrity, no read integrity could be specified. The step of calling SMLS could be skipped and make it faster to complete the request, which has the effect of not having read integrity. Let's say the client is going to have a Get request with key one out there on this dataset. Some where out there on that data component exists key one. In this example, direct request is assumed, which means the index of the VSAM dataset will be searched to find this key. The key will be located in the data component and then returned. Another assumption is that it is an asynchronous request, which means the request will be run on an SRB. You can run request asynchronously or synchronously. Depending on what you specify, a corresponding unit of work is it going to dispatch. For asynchronous request, it means that it will runs on SRB. If you're evaluating CPU time associated with this request, it's going to be associated with SRB time and it's also going to be associated with the client space, not SMSVSAM. Those are the important factors to determine who distributes to the cost.

This client application is running a direct request. So, VRM gets control to handle this request. VRM is going to tell that this is an RLS request and a Program Call or PC is performed to go over to SMSVSAM address space. In SMSVSAM address space, the first thing to be performed is to do a top-down search of the index trying to find this key and trying to figure out where in the data component is this key located.
The index structure is searched through. The number of buffers need to be obtained for the index depends on the number of levels of index. BMF is called to locate the index CI. If they're not in one of those buffer pools up there, then SCM is called to go to the coupling facility. If they're not out there in the coupling facility, then the index component on DASD is searched and read them in. That's where a lot of time spent. The time is already in the hundredth of a second now on doing I/O. So once the index buffers are located, they are searched to find the index CI by using the index buffer, which data CA this key is in. So once that is figured out, that particular CI is looked for in the data component. Before doing that, to the record must be locked since this is a consistent read request. So, SMLS is called to get a record lock. SMLS has to talk to XCF, and goes over to that lock structure on the left to get a shared lock held for this particular record. Once completed, the data CI can be located. The same kind of steps will be performed. A search for the CI will be performed in the buffer pool, if it's not there, SCM will be called to check the cache structure. If it's not there, DASD will be searched. So once that data CI is obtained, control is returned to the client application. The whole point here is to find key one for this particular application, and at the very end, the record must be unlocked. So, that's the whole I/O path.



This slide is an example of processor time for Get request. This shows you the two kinds of the extreme cases. The first case is if Get request was performed and the CI index and the data CI in the local buffer pool were found. In this scenario, this request will run in the thousandths of a second. This is a huge difference. You can reduce one hundred time better in performance if you can get your buffer hit. When it comes to reducing the I/O time, the focus is on the elapse time, not processor time . There will be the same or potentially far more processor time even if you get a buffer hit, but elapse time is where you get the big saving.



Now for a look at those performance parameters that you might specify to setup your environment so that you can try to get a better hit ratio and get better performance. At the system level, RLS keeps all the parameters in SYS1.PARMLIB(IGDSMSxx) member of PARMLIB. Some parameters related to performance are listed in blue. The first one is RLSINIT(YES). It is needed to get RLS to even initialize. The rest are parameters related to performance. The first one is RLS_MAX_POOL_SIZE, which controls the size of the 31bit pool. RLS was initially shipped with a 31bit pool. So there is just this parameter. And, the default to the size of the pool is 100 MB. That parameter is not a hard-coded parameter that the buffer manager must follow. It is just the goal that RLS has to try to adhere to. For example with a 100 MB pool, RLS will try to keep the value to that threshold. However, if RLS needs more than 100 MB buffer at any one time, it will do so. RLS will achieve this value and potentially even go up to that 1.7 gigabyte line. This parameter tells RLS that if this number is exceeded, RLS will work really hard to get that number back down as soon as possible.

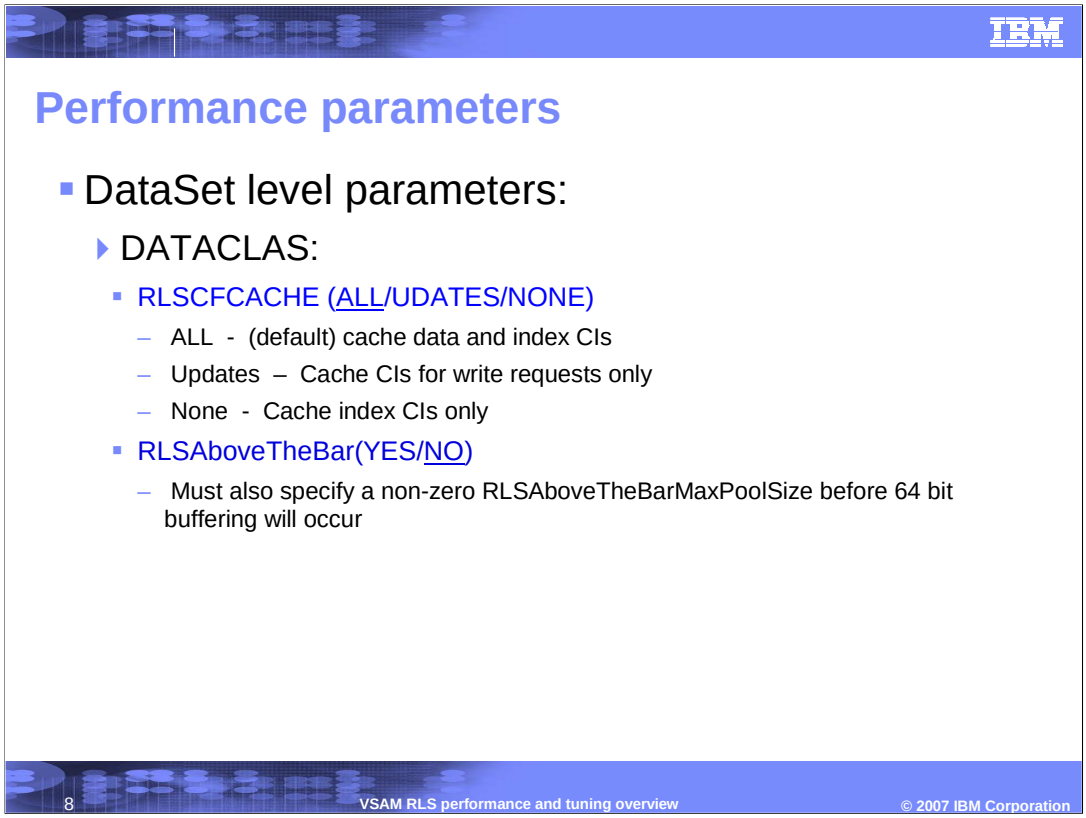
Additional parameters are added to support the 64bit buffering in z/OS® V1R7. There are four parameters. The first one RLSAboveTheBarMaxPoolSize is just like the parameter above for the 31bit pool but this is for the 64 bit pool. So this is again the goal size that the RLU supposed to maintain for the 64 bit pool. The parameter RLS_MAX_POOL_SIZE is a sysplex wide parameter. The first system doing initialize will set it for all systems in the sysplex. You can't have different RLS_MAX_POOL_SIZE for each system in the sysplex. They all have to be the same. However, with a 64 bit pool, each system is allowed to have different sizes. So you can set specify system one is going to have a 4 gigabyte pool; system 2 is going to have an eight gigabyte pool, and so forth. You can use the next parameter RLSAboveTheBarMaxPoolSize to be set to ALL, and all systems will have the same value.

The next parameter RLSFixedPoolSize sustains to both pools: the 31 and 64 bit pool. This parameter page-fixed a certain amount of space for you to specify any value. Those buffers is going to be page fixed. Now, why do you want a page fixed buffer? If buffer are paged and if the LRU wakes up and find out that those buffers are paged out, they are going to be released immediately. So if you have some critical data, and you worried that it might get paged out, those buffers can also be page-fixed. They won't be paged out, and hence released by the LRU.

The next parameter is the CF_TIME and SMF_TIME. These parameters work together. These are the parameters that control the interval in which the SMF 42 record is going to get cut. If you want your SMF 42 record to align with other SMF data that is getting generated by other components, then you would want to specify SMF_TIME equals YES. It's probably the best way to operate so that every other records are getting cut on the same interval.

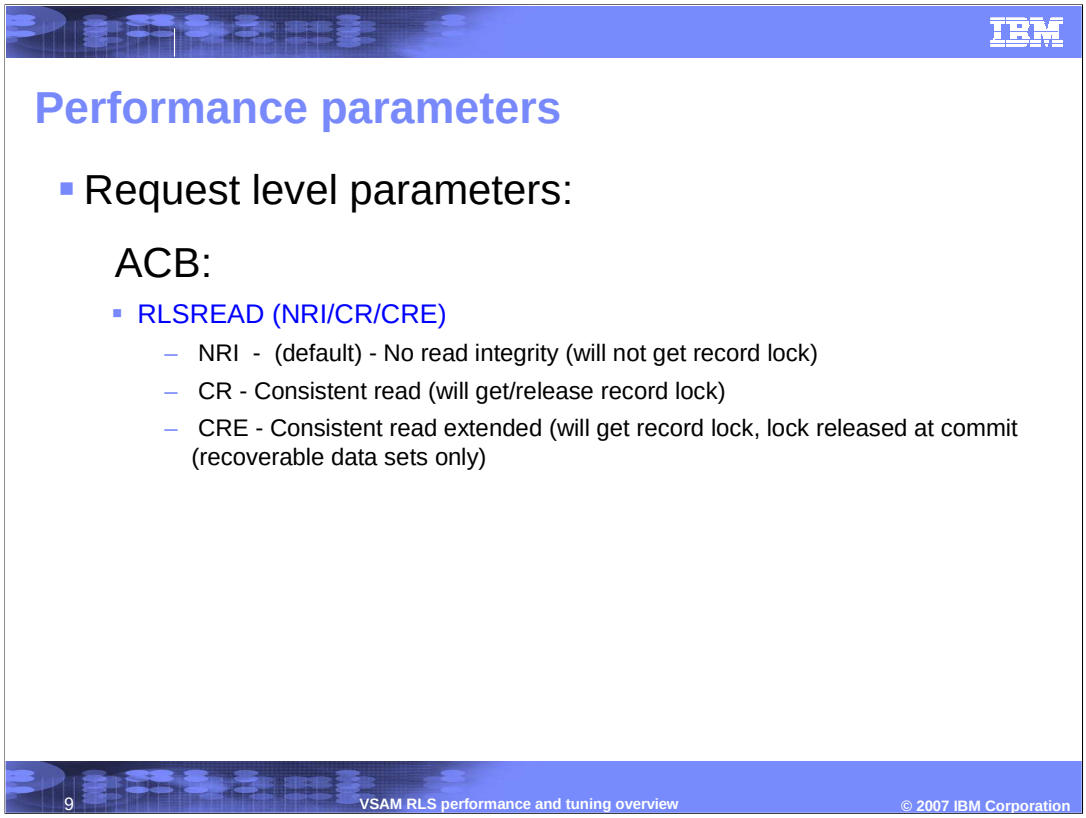
The last parameter RLS_MaxCFFeatureLevel controls the buffer size allowed to go into cache. When RLS was first shipped, the maximum size of buffer that you can put into the CF cache is 4K or less. Any CI with size greater than 4K could not go into the cache. The CI will be read from DASD to the buffer pool directly. With the z/OS 1.4, there is a support for the ability to cache CI greater than 4K and up to the largest CI size of VSAM data set, which is 32 K. The migration of this new feature of being able to cache greater than 4 K is controlled. A parameter is used to specify whether you want to cache CI greater than 4 K.

With Feature Level Z, which is the default, you only cache less than 4 K buffer. If you change to Feature Level A in z/OS 1.4 and above, then RLS will cache CI greater than 4 K in the cache structure.



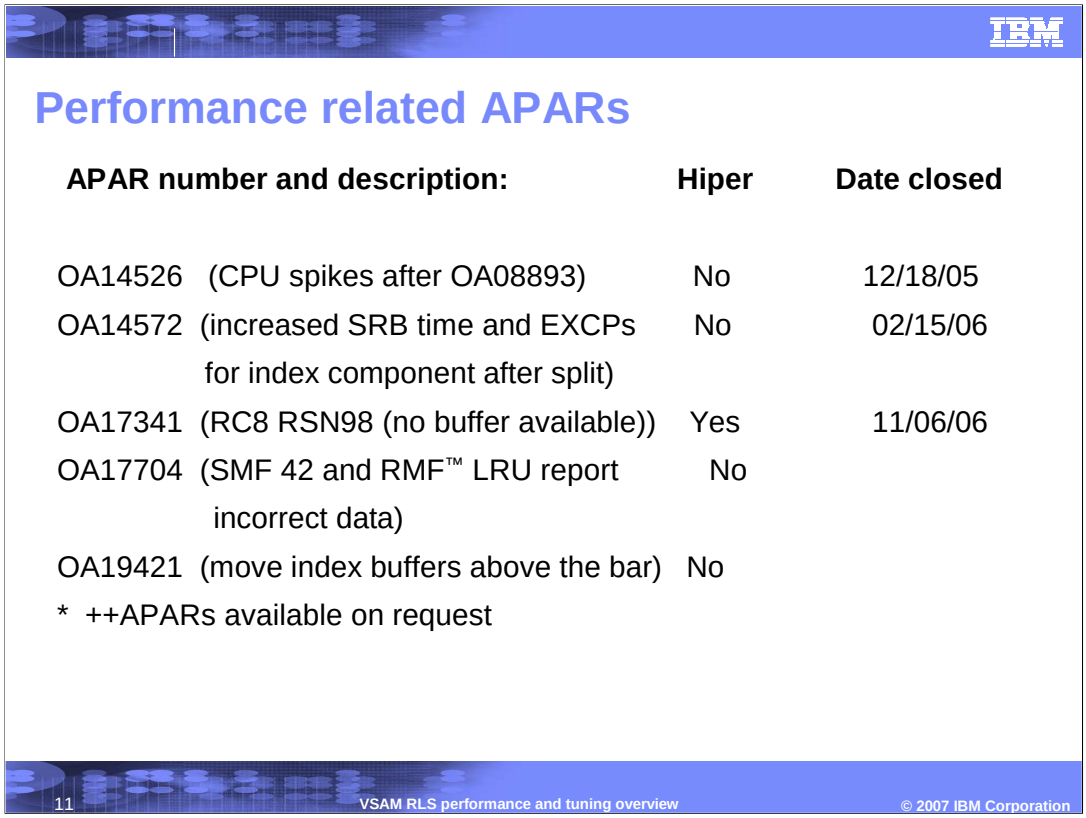
Those are the sysplex wide and system wide level performance parameters. These are the dataset level parameters, which you would specify in the data class. There are two parameters. The first one RLSCFCACHE can be set to ALL, UPDATES, or NONE, with ALL as the default. ALL tells RLS to cache in the cache structure in the coupling facility both the data and index CI, which is a normal case. CFfeature must be specified. This dataset for example you have feature level Z, which indicates only cache 4K or less. Assuming that you have that feature level set, so that dataset can be cached in the coupling facility. You can control which part of the data set you want in the cache. For instance, you can either specify all of the data set or only CI that are being updated, or only the indexed CI should be cached. The index CI is more important than the data CI, because as demonstrated in the GET example. A direct request always requires a top down search of the index. The index CI is always searched. Therefore, index CI has higher priority than data CI. To conserve space in the coupling facility, index CI should be kept in the cache rather than data CI.

The other parameter is RLSAboveTheBar. This parameter specifies either the 31 bit pool or the 61 bit pool to be used for buffering. The default is NO, which means the dataset is going to the 31 bit pool. If you specify YES, then this dataset is going to the 64 bit pool.



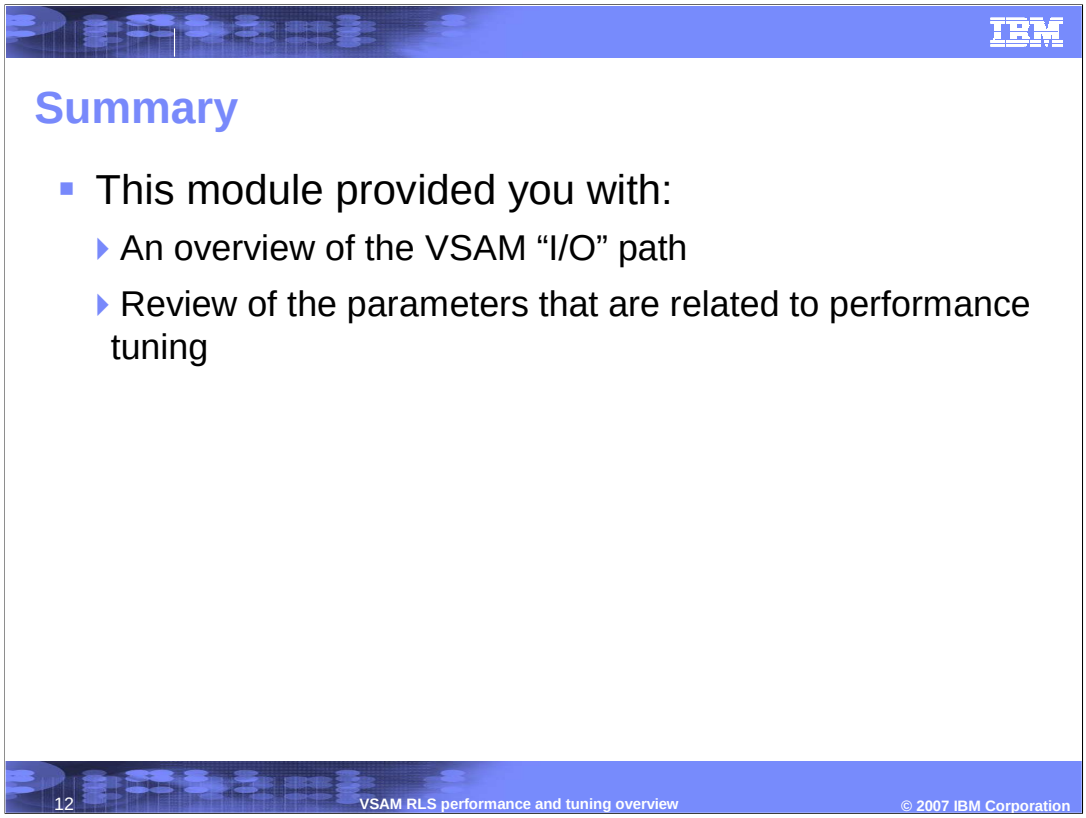
Next, there are parameters at the request level. When you are issuing your GET and PUT, or when you open a data set, these options would affect the performance. The first one is the RLSREAD parameter, which specifies read integrity type. The value NRI indicates no read integrity, which tells RLS to skip the step of having SMLS to get a record lock. The other two options Consistent Read or Consistent Read Extended will go to get the record lock. If you don't care about read integrity, so don't waste time to get a record lock and specify NRI to improve performance.

At the RPL level, the options you would put on the RPL request can be either asynchronous or synchronous. It shows you the kind of unit of work you are going to run under and how you might measure that. With asynchronous requests from performance point of view are better because SRBs cannot be interrupted all the time. They get to run to completion. Unless they're going to suspend, then they'll get interrupted. If You want your request to run quicker, then running asynchronous request is better. Some of the new parameters have impact on performance, whether you're going to read direct or sequential. For a direct request, RLS constantly searches the index to find the data CI. That is more costly than a sequential read or horizontally read through the index. So if you're trying to read a lot of records quickly, then sequential is a better way. You can also specify the read option on the RPL as well when you open a dataset.



There are some available APARs related to performance. Some of them fix bug related issues and some of them are enhancements related. The APARs shown here are recommended to be on your systems before you are serious about getting into the 64 bit buffering or performance in general. The first four are bugs related. The last one is enhancement, OA19421. The 64 bit design only has data buffers in the 64 bit pool. So even if you said RLSAboveTheBar(YES) for your data set, only the data buffers will be bufferred in the 64 bit pool. The index buffers are still staying in the 31 bit pool. With this APAR OA19421, the index buffers are also moved into the 64 bit pool.

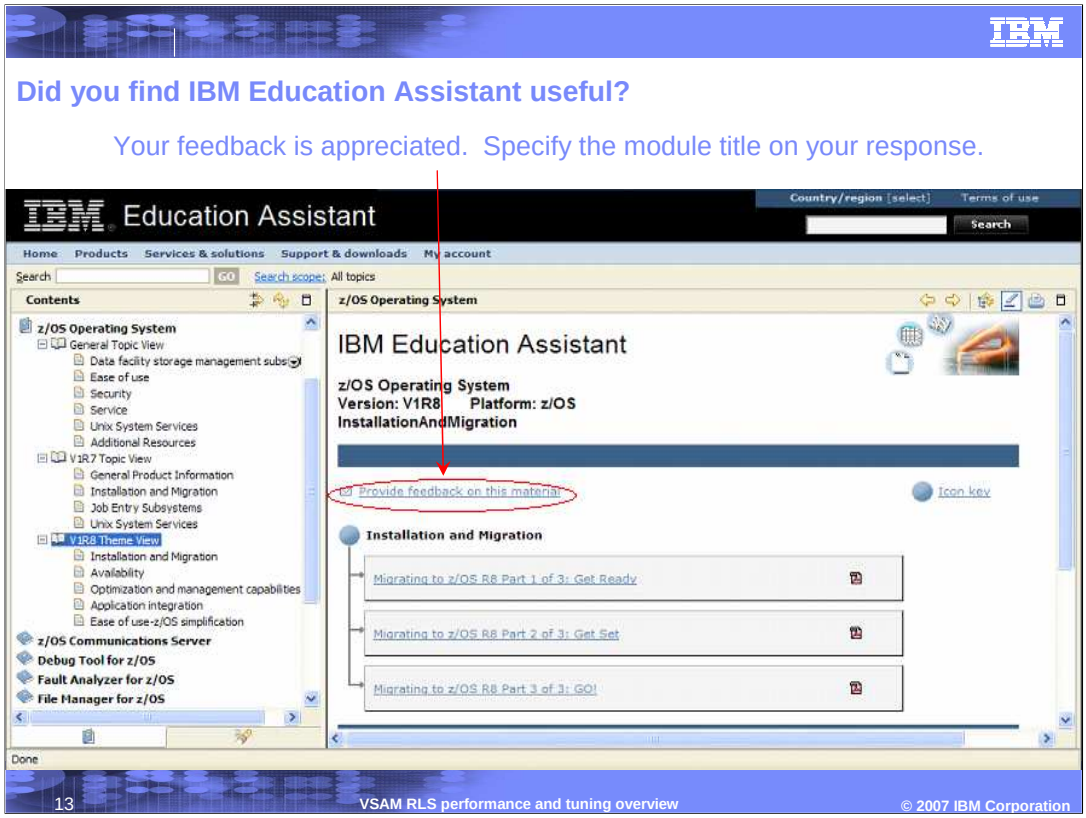
Check with the support center on the status of APAR closing.



In order to talk about the VSAM RLS I/O path, you need to understand the RLS component that makes that path. This presentation provided you with some of the terminologies that you will need to use. A specific example of the I/O path was included. This presentation focused on the GET path, because the GET path clearly shows you where you can get your benefit on performance.

There are various parameters at different levels. There are sysplex or system parameters that you can set. There are dataset level parameters and other parameters that are even at the request level that you can set that help control the performance and tuning.

Check out the IBM Education Assistant module to obtain more information on the setting of dataset level parameters and other parameters that are even at the request level that you can set that help control the performance and tuning.



- **In order to supply you with pertinent and timely information in IBM Education modules, your opinions are important. To help IBM in creating these modules, take the time to help us out. In your feedback to IBM please answer the following three questions:**

1. **How helpful was this IEA presentation? Give a rating from 1 to 5 where 1 = very helpful and 5 = not at all helpful.**
2. **Did this presentation save you a service call to IBM? Yes or No.**

- **3. If there are any other topics you would like to see covered in IEA, what are they? _____**

Trademarks, copyrights, and disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM RMF z/OS

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements or changes in the products or programs described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (for example, IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.