IBM z/OS 3.1

AI-powered WLM batch initiator management

Resource and Tuning Guidelines

Contributors:
Joseph Gentile, IBM
Loic Fura, IBM
Paul Ippolito, IBM
Tian Na, IBM
Sun Yu Zhuo, IBM
Khadija Souissi, IBM
Robert St. John, IBM
Julia Bulgannawar, IBM
Nicholas Matsakis, IBM
Dr. Paul Lekkas, IBM
Chris Watson, IBM
Steve Partlow, IBM
Frank Bellacicco, IBM
Terri Menendez, IBM
Robert Miller Jr, IBM

Content

Preface

This paper is intended to provide guidance for planning and implementation of the AI-powered WLM batch initiator management offering. The document explains the solution at a high level, introduces some of the major components and how they work together, and ties those to the resources that you will need to configure or set aside for the solution. The information is based on internal testing, projections, and subject matter expertise. The authors intend to help users like system programmers understand the resources used by the solution more thoroughly before implementing it themselves and providing them with a more seamless experience.

1. Introduction

IBM z/OS® 3.1 introduces the AI Framework for IBM z/OS, or z/OS AI framework for short, which is a set of AI capabilities provided with the operating system and designed to incorporate and operationalize pre-built AI models geared towards simplifying system operations and IT processes.

AI-powered WLM batch initiator management represents the initial AI infused capability within the operating system that leverages the AI Framework for IBM z/OS. It is designed to intelligently predict upcoming batch workload and react accordingly. As a result, batch workload spikes can be forecasted and addressed right away, helping system programmers avoid fine tuning and trial and error approaches. You can leverage the new AI capabilities infused into WLM without any need for additional AI skills.

Using an intuitive control interface, called AI Control Interface for IBM z/OS, you can switch between AI, non-AI, and simulating modes. Simulating mode allows you to extract predictive insights from the model without the system taking automated action. It is intended to help you gain confidence and trust in AI mode before enabling it.

In the next section, you will find a high-level overview of each of the components and how they interact with each other, followed by a description of the environment used for testing. Section 2 describes the hardware resource requirements and utilization for both training and inferencing use cases. Section 3 addresses software tuning guidance. Finally, section 4 describes the resource requirements and performance tradeoffs of model training.

1.1 High Level Offering Components and their Interactions

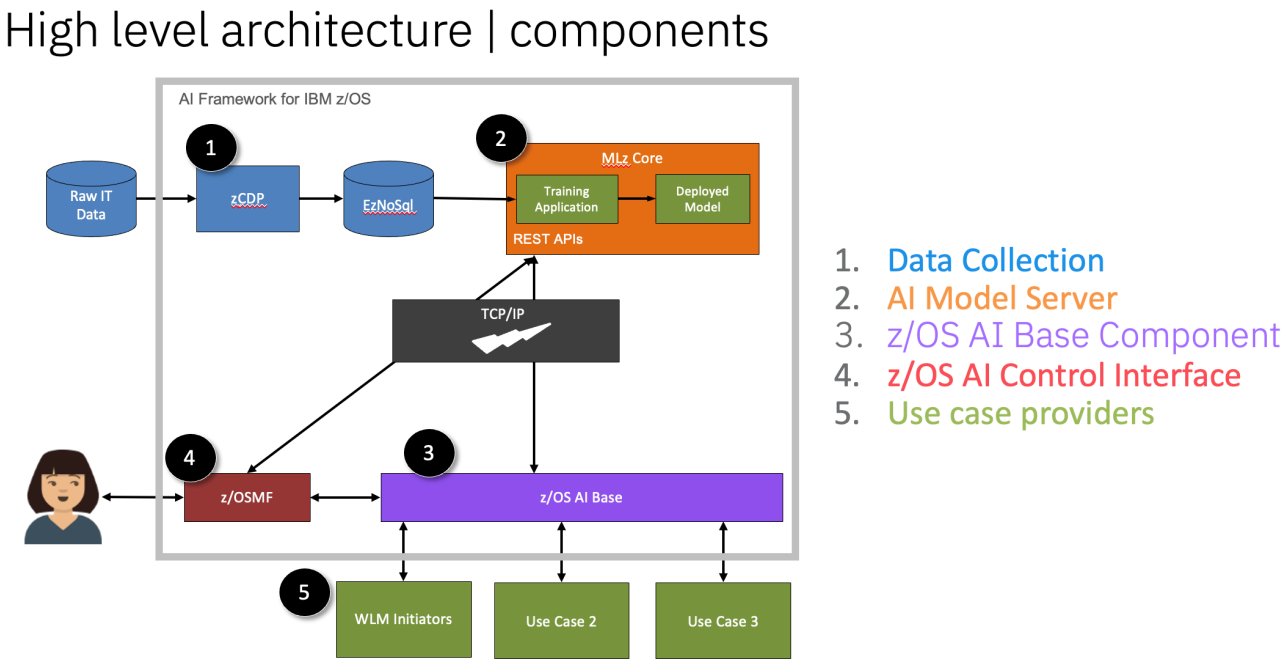
The major components of the AI framework are:

- AI Base component for IBM z/OS, a new operating system component
- EzNoSQL for z/OS
- IBM Z® Common Data Provider
- Machine Learning for IBM z/OS Core
- The AI Control Interface for IBM z/OS, a new z/OSMF plug-in

While most of the components are provided within IBM z/OS, IBM Z Common Data Provider and Machine Learning for z/OS Core are delivered within the AI System Services for IBM z/OS. This new offering integrates seamlessly with the other z/OS AI Framework components and delivers support of key AI lifecycle phases including data ingestion, AI model training, inference, AI model quality monitoring, and retraining services.

Figure 1 shows the solution architecture picture including the z/OS AI Framework components, the initial use case provider as well as place holders for future use case providers.

Figure 1:



Following is a components description including how they interact with each other.

- z/OS Workload Management (WLM) is the initial use case provider that leverages the AI Framework for IBM z/OS.
  - For systems configured to use AI, Workload Management queries the AI model on a 10-second interval via AI Base component for IBM z/OS services, for participating service classes in a single request, to predict the upcoming workload based on historical workload patterns and start the required initiators in advance.
  - WLM writes SMF 99 subtype 2 data to SMF which is used as the historical data, which contains data for any service classes that had recent activity.
  - The real-time SMF records are used during the inference requests to predict the upcoming workload.
  - WLM provides an AI training application which can be trained with your own historical SMF 99 subtype 2 records. The resulting trained AI model is deployed automatically in the model server which is leveraging Machine Learning for IBM z/OS Core.
  - The training takes place at the system level, such that one model is produced for each participating system.
  - Each model knows about all the service classes on the system for which it was trained.
    - For training to be successful for a given service class on a system, sufficient historical data must be provided.
    - At the time of the writing of this paper, the minimum is 30 days.
- z/OS System Management Facility (SMF) – SMF provides the services for recording SMF data. There are two types of SMF recording: Data set (e.g. SYS1.MANA), and the more strategic log stream.
  - SMF allows real time access to records in memory via SMF real time in-memory objects. The types of records that are buffered and the size of the memory object are configurable in SMFPRMxx.
  - IBM Z Common Data provider can leverage the real time interface but logstream recording is required.
  - An advantage of the real time interface is to avoid adding exit routines to SMF exits.
- IBM Z Common Data Provider (zCDP) – This component acts as the vehicle to transport and propagate the data to the datastore in JSON format on a 10 second interval.
  - The data is intended to be consumed by the AI training application or model training or during inferencing.

  - In the specific use case of AI-powered WLM batch initiator management, the data consists of SMF 99 subtype 2 records.
  - zCDP has two started tasks:
    - The System Data Engine (e.g. HBOSDE) filters and collects SMF data from z/OS.
    - The Data Streamer (e.g. HBODS) streams the data to the data store.
  - The System Data Engine supports using both SMF in-memory object or provided SMF exit routine in order to collect the records.
  - zCDP runs on each participating system to collect and stream the historical data to the datastore.
- EzNoSQL for z/OS – EzNoSQL is a z/OS native JSON data store which uses VSAM RLS as the back-end database.
  - The AI Framework for IBM z/OS leverages EzNoSQL to store the historical data.
  - Each AI model and participating system will have its own EzNoSQL data store containing historical data for that system.
  - EzNoSQL provides Java APIs which are used in this solution to access the data store.
  - EzNoSQL uses a VSAM RLS Key Sequenced Data Set (KSDS) to store the data.
- z/OS VSAM record-level sharing (VSAM RLS) – VSAM record-level sharing is an access method for a VSAM cluster which enables Sysplex-wide sharing of the cluster.
  - Data is cached both locally on each system accessing the cluster in the BMF buffer pool space, and in the coupling facility cache structure associated with the cluster.
  - VSAM RLS utilizes the IBM Z coupling facility to provide the global cache and the serialization for the EzNoSQL data store.
  - In this solution, an alternate index is used to allow the AI model to quickly access records by service class and chronologically.
- Machine Learning for IBM z/OS Core (MLz Core) – Machine Learning for IBM z/OS Core plays the role of the AI model server. It provides AI model training, deployment, management, and inferencing capabilities.
  - The scoring service is an address space that contains a runtime environment in which the model can execute inference requests (queries). It is typically called ALNSCSV.
  - Leveraging the pre-built AI training application, you can train the AI Model with one button click leveraging the IBM Apache Spark environment which is incorporated in Machine Learning for IBM z/OS Core.
    - The Spark environment includes Spark master and worker address spaces that are persistent (e.g. ALNSPKM and ALNSPKWx respectively).

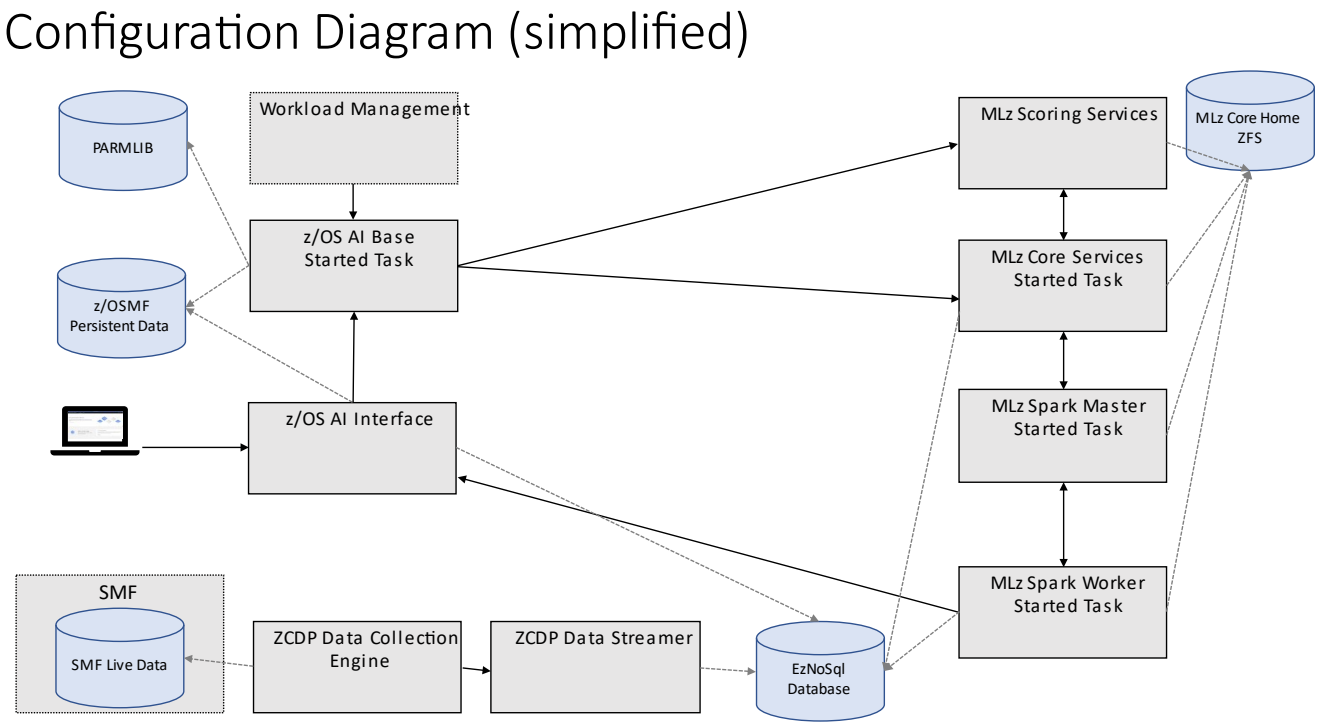
        - During training, Spark creates a driver and one or more executor address spaces in order to fulfill the request (e.g. ALNSPKDx and ALNSPKXx respectively).
    - Machine Learning for IBM z/OS Core also includes a core services address space which receives REST API training or inference requests and routes control to the appropriate backend service to handle them.
    - Only a single instance of Machine Learning for IBM z/OS Core and the scoring service is necessary for the Sysplex.
- AI Base Component for IBM z/OS (z/OS AI Base) – This new z/OS component acts as a bridge between Workload Management and Machine Learning for IBM z/OS.
    - It receives inference calls from Workload Management, forwards them via TCP/IP communication to Machine Learning for IBM z/OS, and the request is passed to the AI model scoring service.
    - The z/OS AI Base component forwards the inference result provided by the scoring service to Workload Management.
- IBM z/OS Management Facility (z/OSMF) – z/OS Management Facility provides the configuration workflows for this solution as well as the AI Control Interface plug-in.
- AI Control Interface for IBM z/OS (z/OS AI Interface) – The AI Control Interface for IBM z/OS allows you to control and interact with the AI capability on the AI configured system. Leveraging this interface, you can initiate the AI model training (or re-training), and toggle AI modes at the service class level. You can select from AI enabled, AI disabled, or AI simulating modes.

Figure 2 (below) shows a slightly lower level diagram of the components and their interactions than Figure 1. They are summarized in three steps:

1) Data collection: in the lower left corner, the SMF Live Data is being collected and converted into JSON by the zCDP data collection engine then sent to the EzNoSQL database using the zCDP data streamer.
2) AI services for scoring and training: in the upper right corner, MLz Core services offer APIs for both management and scoring. The scoring API allows inference requests to be submitted and returns predictive insights. The management API can be used to request model training or deploy a new model. The model training runs as a Spark application under MLz Core managed Spark cluster.

3) Workload management: in the top left corner, the z/OS AI Base component provides access to the AI model services to WLM. WLM interacts with the z/OS AI Base component to request predictive AI insights and adjust its decisions for batch initiator management. Using the new z/OS AI Control Interface in the left center portion of the diagram, system programmers can initiate a model training and select which service classes should be AI managed.

Figure 2:



1.2 Performance Test Environment

The computer system used for testing the solution was an IBM z15® 8561-716 T01 with two internal coupling facility LPARs. The largest z/OS LPAR configuration tested had 1 shared CP and 8 shared zIIPs with 256 GB of memory. All logical processors were vertical high polarity by LPAR weight. The smallest LPAR configuration tested had 1 shared CP and 1 shared zIIP with 32 GB of memory. We used approximately 52 GB of DASD space. For TCP/IP networking, we configured SMC-D. However, the test environment consisted of a single system at the time of writing this paper, so network communication was local only. For most of the testing, 3 WLM managed service classes have been used. Some additional scalability measurements have been taken with up to 8 WLM managed service classes.

Test Environment Resource Name	Value
CPC	IBM z15 8561-716 T01
LPAR	1
Internal Coupling Facility	2
LPAR CPs	1
LPAR zIIPs	Minimum: 1 Maximum: 8
LPAR Memory	Minimum: 32 GB Maximum: 256 GB

Key software stack installed in the performance test environment

The following table is provided for reference to indicate which software levels were tested during this study. It is not an exhaustive list.

It is recommended to stay current on maintenance for each AI Framework for IBM z/OS component and Workload Management. The z/OS Fix Category (FIXCAT) IBM.z/OS.AIFramework can be used to find the related service. For EzNoSQL service, you can use the IBM.Function.EzNoSQL FIXCAT. For more information about Fix Categories, please refer to https://www.ibm.com/support/pages/ibm-fix-category-values-and-descriptions.

Product or z/OS Component Name	APAR	PTF
Pre-release version of z/OS 3.1	N/A	N/A
IBM z/OS WLM	OA64632	UJ93376
	OA64643	UJ93364
	OA65359	UJ93761
AI Base Component for IBM z/OS	OA64497	UJ93388
AI Control Interface for IBM z/OS	PH53888	UI92846
	PH53940	UI93018
	PH56159 pre-release version	N/A
EzNoSQL for z/OS	OA64811	UJ93294
	OA63987	UJ93161
	OA64239	UJ93410
IBM z/OS JES2	OA65199	UJ93426
IBM z/OS Data Gatherer	OA64631	UJ93229
Pre-release version of Machine Learning for IBM z/OS Core 3.1	N/A	N/A
Pre-release version of IBM Z Common Data Provider V5.1	N/A	N/A

2. Hardware Resource Guidance

The primary CPC resources required for the offering are memory (central storage), zIIP processors, and at least one coupling facility. The sections below describe these topics in detail.

2.1 Memory Resource Guidance

The AI model uses memory during inferencing and training. The pre-built AI training application has been developed using the Scala programming language and is compiled to a Java executable. Java™ can use up to the maximum heap size allowed, which is configured in the AI Framework Configuration Workflow. At the time of writing this paper, the default setting for the MLz Core scoring service Java Virtual Machine (JVM) Heap Size is 64 GB of virtual memory which actually configures the –Xmx Java heap option. In our testing, we found out that this value was larger than the amount that was used. Therefore, we recommend setting the value to the minimum allowed in the workflow, which is 16 GB. This should allow you to reduce the potential memory footprint of the scoring service and CPU impact of Java garbage collections.

If you want to change the settings afterwards, please refer to the instructions below to modify the Java heap settings directly in the scoring service configuration file. However, most users should not need to make any change to it, except if trying to reduce the memory footprint to its minimum. In our testing environment, it typically consumed under 1 GB of heap memory with the Java heap size set to 1 GB minimum (-Xms) and 2 GB maximum (-Xmx).

> How to change the Java heap size for the MLz Core scoring service:
> 1) Open $IML_HOME/configuration/scoring.cfg.[scoring_service_name] where [scoring_service_name] is the name you have selected for the scoring service.
> 2) Locate the “jvm_options” setting.
> 3) Update the “-Xms” value to set the minimum heap size to the desired value.
> 4) Update the “-Xmx” value to set the maximum heap size to the desired value.
> 5) The change becomes effective when you (re)start the scoring service.

Training the model may require 16 GB of virtual memory, depending on how Spark is configured. See section 4.2.3 for a discussion of training memory consumption.

z/OSMF may require 1 GB of heap memory depending on configuration. IBM Z Common Data Provider may require up to 1 GB of heap for the data streamer.

Furthermore, you need to consider the ZFS user file cache size. In our testing environment, we used the system-calculated value of 2 GB. For more information, please refer to section 3.2.1.

If SMF in-memory objects are used, you need to consider the size of the object. In our testing we employed a 500MB object.

Be sure to consider the VSAM RLS BMF buffer pool size as well, especially if it is expanded for this solution. In our testing, we found out that model training tended to use a lot of buffer pool space, as all the historical data is ingested by the training application. Inferencing alone did not require much buffer pool space. Please refer to the section 3.1.1 to learn more about tuning guidance for the VSAM RLS BMF buffer pools.

Real memory demand is typically lower than virtual memory demand. However, these components alone can theoretically consume these amounts of memory. When measuring model inferencing, the RMF high-water mark system memory demand was 8.8 GB. When measuring an 8-service class model training, the high water mark system memory demand was 11.2 GB excluding VSAM RLS BMF buffer space.

We calculated the high water mark system memory demand based on the SMF 71 RMF Paging Activity Report central storage numbers, using the following formula:

High water mark frames in use = Total available frames – Minimum available frames

The high water mark frames in use can be converted to gigabytes by multiplying the value by 4096 bytes per page and then dividing it by 1024 cubed (1 GB).

$$Gigabytes\ of\ real\ memory\ = \frac{Frames \times 4096}{1024^3}$$

The table below summarizes these figures and adds an address space level breakdown for the top 64-bit memory consuming address spaces. We calculated the address space level breakdowns using SMF30HVR (above the bar real frames used) and the same formula above to convert frames to gigabytes.

There is a discrepancy of roughly 2 GB between the sum of the SMF 30 64-bit Real Memory numbers and the RMF high water mark numbers, which is presumed to be below the bar real memory. By and large, this solution exploits above the bar memory.

If the memory is not available in central storage, then the operating system must page to free up the required memory which impacts system performance. Our intention is to avoid paging

for better performance. We recommend at least 32 GB as a starting point which has room for expansion. Please keep in mind the memory demands of your workload today, though. During

the time where we tested with 32 GB of memory, the test system did not have other workloads running which consumed much memory. We advise to always monitor your systems for memory shortages and paging especially when adding new software products or workloads.

Scope	Memory utilization during inferencing	Memory utilization during training	Data Source
High water mark System memory demand excluding VSAM RLS Buffer Pools	8.8 GB	11.2 GB	RMF Monitor I Paging Activity Report (SMF 71)
ZFS with 2 GB User File Cache	1.9 GB	2.3 GB	SMF 30 subtype 2 field SMF30HVR
ALNSPKD0 - MLz Core Spark Driver	N/A	1.7 GB	SMF 30 subtype 2 field SMF30HVR
ALNSPKX0 - MLz Core Spark Executor	N/A	2.2 GB	SMF 30 subtype 2 field SMF30HVR
ALNSCSV – MLz Core Scoring Service	1.5 GB	1.5 GB	SMF 30 subtype 2 field SMF30HVR
IZUSVR1 - z/OSMF Server	510 MB	550 MB	SMF 30 subtype 2 field SMF30HVR
ALNSERV – MLz Core REST Server	200 MB	300 MB	SMF 30 subtype 2 field SMF30HVR
ALNSPKM – MLz Spark Master	110 MB	170 MB	SMF 30 subtype 2 field SMF30HVR
ALNSPKW – MLz Core Spark Worker	120 MB	160 MB	SMF 30 subtype 2 field SMF30HVR
HBODS – zCDP Data Streamer	90 MB	110 MB	SMF 30 subtype 2 field SMF30HVR

2.2 Processor Resource Guidance

From a processor perspective, a zIIP is not required as CPs can be used, but the AI model processing is highly zIIP eligible. In our testing, the CP utilization during inferencing was negligible and the zIIP utilization was mid-single digit percentages in terms of one processor. The zIIP utilization for training is much more important because AI model training is processor intensive. During training, Spark can use 100% of any zIIP it is allowed to use according to the Spark configuration. There is a tradeoff between training elapsed time and zIIP processor resource which is discussed in detail in section 4 on training.

For most of the testing, we have not used Simultaneous Multithreading (SMT) for zIIP. We found out that SMT did not have much of an impact on the performance during testing. Please refer to section 4 to learn more about leveraging SMT for training.

You can find the information regarding the processor resource recommendations and utilization in the table below. The data is based on testing with a number of service classes between three and eight.

Resource	Minimum	Recommended (if different)	Average utilization during inferencing	Average utilization during training
zIIPs (shared)	1	2+ for training parallelism (see section 4)	2-5% of one processor	Up to 100% of all allowed zIIPs
CPs (shared)	1 (at least one is required for z/OS)		< 2% of one processor	~5% of one processor

2.3 DASD Resource Guidance

The major file systems and data sets for the solution are summarized in the following table, along with how much DASD space we allocated for each in our test environment.

Item	DASD Space
MLz Install, Home, and Spark Home filesystem	47.4 GB (this is what we tested with but a minimum of 10 GB is required)
MLz setup user file system	500 MB
zCDP setup user file system	300 MB
EzNoSQL for z/OS	4 GB

3. z/OS System and AI Framework for IBM z/OS Tuning Guidance

This section describes operating system and AI framework tuning considerations with the goal of optimizing the AI framework. We organized the section by product/component.

3.1 EzNoSQL and VSAM/RLS Tuning Guidance

3.1.1 IGDSMSxx PARMLIB Settings

Since EzNoSQL relies on VSAM RLS, it is important to update and set IGDSMSxx on your system with an adequate BMF buffer pool size. This ensures that VSAM RLS can cache data from the file in memory for low latency access. The above the bar option, RlsAboveTheBarMaxPoolSize, allows for 64-bit buffer pools that are larger than the maximum size of the below the bar pool of 1.7 GB. The minimum size allowed for the above the bar pool is 500 MB. Using the above the bar pool will facilitate local caching of a large amount of VSAM data. For this offering the VSAM data size is expected to be multiple gigabytes. Other VSAM RLS exploiters should be taken into consideration as well when sizing the buffer pools because they are a shared system resource.

It can also be advantageous to page fix a portion of the buffer pools with the RlsFixedPoolSize setting to reduce CPU utilization associated with VSAM RLS I/O. The fixed buffer pool can avoid the need for memory to be fixed and unfixed on every I/O. However, once one of the fixed buffers are designated to a certain CI Size like 4K, it is permanently fixed at that size, at least until the next time SMSVSAM is restarted. That means the buffer would be ineligible for use by a VSAM file CI of a different size like 32K.

Consider that the buffers are shared among all VSAM RLS data sets accessed by the system. Therefore, we recommend to avoid fixing too large of a percentage of the total buffer pool

size. If the buffer pool space is sufficiently large, for example tens of gigabytes, then a larger percentage can be safely fixed. If the buffer pool space is less than 10 GB, then a smaller percentage should be fixed. For such a buffer pool, a good starting point may be to fix 20% of the buffer pool size.

In general, we recommend sizing the VSAM RLS BMF buffer pool so that it will achieve above a 90% or higher BMF buffer pool hit rate. During inferencing, we have noticed that it is possible to achieve a buffer hit rate above 90% with a buffer pool that is a fraction of the data size.

During training, it may not be possible to achieve a 90% buffer hit rate because the training application reads all the historical data from the datastore. However, sizing the buffer to achieve the maximum hit rate for training may reduce I/Os during training and the associated CP utilization. Since model training only takes place occasionally, this decision will also depend on how much memory you have available to dedicate to VSAM RLS BMF buffer pools.

To optimize your buffer pool size for inferencing alone, 20% of the data size including the alternate index (AIX®) may be a good starting point. To go further and optimize for training, we recommend starting at 100% of the data and AIX size. These values should be tuned by monitoring the buffer pool hit rate, though.

You can monitor the VSAM RLS BMF buffer pool hit rate, Coupling facility (CF) cache structure hit rate, and DASD rate using RMF™ Monitor III or similar product or SMF 42 subtypes 15-19. Data set level statistics are available via RMF Monitor III or similar product or enabling the SMF 42 subtype 16 records. This granular monitoring capability can help you measure performance specifically for your EzNoSQL database.

RLS_MaxCfFeatureLevel=A is recommended to allow data caching greater than 4K in the CF.

These settings can be dynamically activated with the following MVS command: SET SMS

You can use the following MVS command to confirm the settings have taken effect: DISPLAY SMS,OPTIONS

3.1.2 SMS DATACLAS Settings

When configuring EzNoSQL for this solution, it is important to use a DATACLAS that has the following attributes. Extended addressability allows a data set to be larger than 4 GB, which is

required for the service class history datastore. RLS Above the Bar is recommended to allow VSAM RLS to use the larger above the bar buffer pools.

RLS CF Cache setting DIRONLY (directory only) prevents VSAM RLS from caching data in the CF cache structure for this cluster. The structure is still used to access the directory that indicates which medium has the latest copy of the data (e.g. buffer or DASD). DIRONLY is recommended to minimize CPU utilization associated with accessing the Coupling facility. This was the option we used during the test since our environment was single system. If more than one system is participating, then there is likely value to cache data in the CF for inferencing and training, but it may also increase CPU utilization.

RLS CF Cache can also be set to ALL, NONE, or UPDATESONLY. ALL will cache reads and writes in the Coupling facility cache structure. NONE will only cache VSAM index data. UPDATESONLY will cache only write requests. Please refer to the following to learn more about these options
https://www.ibm.com/docs/en/zos/3.1.0?topic=sharing-record-level-cf-caching.

DATACLAS recommendations are summarized in this table for convenience.

SMS DATCLAS Setting Name	Setting Value	Context
Extended Addressability	Yes	• Greater than 4 GB data sets
RLS CF Cache	DIRONLY	• Reduce CPU cost from CF I/O
RLS Above the Bar	Yes	• Allow above the 2G bar memory to back local buffer pools

To change these settings, you can use the ISMF application on ISPF. Select option 4 and scroll to the right with PF11:

```
LINE       DATACLAS LAST TIME                        EXTENDED
 OPERATOR   NAME     MODIFIED   DATA SET NAME TYPE   ADDRESSABILITY
---(1)----  --(2)--- --(25)---  -------(26)-------   -----(27)-----
            DCZHYPL  03:21      EXTENDED REQUIRED    YES

. . .
 LINE       DATACLAS RLS CF        EXT CON   RLS ABOVE
 OPERATOR   NAME     CACHE         REMOVAL   THE BAR
---(1)----  --(2)--- ---(43)----   -(44)--   --(45)---
            DCZHYPL  DIRONLY       NO        YES
```

3.1.3 CFRM Policy Settings

For EzNoSQL, we recommend some CFRM Policy settings to optimize VSAM RLS CF lock and cache structure performance for this solution.

First, we recommend not to enable duplexing for any VSAM RLS lock structure such as IGWLOCK00. Duplexing adds significant overhead to structure accesses in these cases and is not recommended for VSAM RLS.

In general, it is recommended to use separate lock structures for different VSAM RLS exploiters, for instance recoverable applications, for VSAM RLS availability. Recoverable applications have the potential to hold a large number of locks concurrently and may impact lock structure availability for EzNoSQL and other exploiters that may be sharing the structure. Using separate lock structures for different exploiters prevents them from impacting each other.

We also recommend dedicating a new cache structure for this solution, so that it does not have to compete with other VSAM RLS exploiters, reducing cache invalidations and contention. For structure sizing guidance, consult the CF Sizer tool, and select the VSAM RLS page. Please use the following link to the tool: https://www.ibm.com/support/pages/cfsizer. This tool assumes you are caching data in the Coupling facility via RLS CF Cache DATACLAS setting (above). If you are using DIRONLY, you may be able to size the structure smaller than this value. In our testing, in conjunction with DIRONLY, we used a size of 300 MB and saw negligible impact. We don’t recommend defining a VSAM RLS cache structure size below 100 MB.

For VSAM RLS lock structures or any lock structures, it is important to monitor the false contention rate. If the false contention rate is too high, for instance greater than

0.5%, then you can generally reduce it by increasing the structure size. The CF Sizer tool can also assist with sizing your VSAM RLS lock structure.

It is also good to keep an eye on the asynchronous request rate for both lock structures and cache structures associated with VSAM RLS. A rise in asynchronous requests could indicate that the structure is allocated in a CF that is too busy from a utilization standpoint, too far away from the CPC for the requests to complete synchronously, or that some other bottleneck is present.

You can monitor both false contention and asynchronous requests using RMF or a similar product, or the data from SMF 74 subtype 4 Coupling Facility Activity.

Finally, it is noteworthy that latency is lowest when the structures are allocated on a CF LPAR co-located on the same CPC as the z/OS LPAR. The structure allocation PREFLIST keyword can be used to control this. For Sysplex environments spread across multiple CPCs, this may not be possible. You can use the following MVS™ command to query the CFRM policy settings and status for a structure:

```
DISPLAY XCF,STR,STRNAME=structurename
```

For more detailed VSAM RLS guidance, consult the “VSAM RLS Performance and Tuning Presentation” document linked here: https://www.ibm.com/support/pages/node/319659.

CFRM Policy Setting	Context
VSAM RLS lock structures such as IGWLOCK00 Duplex DISABLED	• Minimize CF communication overhead
Separate VSAM RLS lock structure for different VSAM RLS exploiters	• VSAM RLS Availability
EZNOSQL_CACHE str dedicated for this use case	• Isolate data for this offering from other VSAM RLS data to avoid impacting caching performance
Monitor false contention for lock structures and asynchronous request rates using performance monitoring tool or SMF 74 subtype 4	• Minimize CF communication overhead
EzNoSQL cache and list structures allocated on local CF LPAR (PREFLIST) if possible	• Minimize CF communication overhead

3.2 Miscellaneous Tuning Guidance

3.2.1 ZFS

As the solution is OMVS based, you should consider ZFS settings. First, the user file cache should have a low fault rate of 10% or lower which means that the hit rate should be 90% or higher. The default user file cache size is 10% of LPAR memory up to 2 GB or 256 MB if 10% of real is less than 256 MB. In our testing, we found out that for this single use case, 2 GB may be a good starting point. This cache can be as large as 64 GB.

Secondly, the Vnode cache hit rate should be above 90%. The Vnode cache is a sort of file directory cache in ZFS. In our testing, we found the system default of vnode_cache_size = 32786 vnodes may be a good starting point.

If the memory is available, we recommend specifying the fixed option for each of the caches to reduce the CPU cost associated with each ZFS I/O. Note that fixing the caches permanently occupies real memory.

You can adjust these settings in the ZFS configuration file IOEFSPRM or PARMLIB member IOEPRMxx. Be sure to check out the documentation link for more information https://www.ibm.com/docs/en/zos/3.1.0?topic=ioefsprm-.

Keep in mind the ZFS settings are system wide, and therefore must accommodate all applicable workloads.

In order to query the cache hit rates, you can issue the following MVS command and consult the output: `MODIFY ZFS,QUERY,ALL`
First it is recommended to clear the stats with the following MVS command, and then run your workload: `MODIFY ZFS,RESET,ALL`
Afterward, issue the following command to check the statistics: `MODIFY ZFS,QUERY,ALL`

More information can be found in the Performance Tuning chapter of z/OS File System Administration:
https://www.ibm.com/docs/en/zos/3.1.0?topic=debugging-performance-tuning.

3.2.2 Component Trace (CTRACE)

Component trace options can influence CPU utilization. For this offering and in general, we recommend for best performance that component trace (CTRACE) should collect minimum

information for RSM, OMVS, and other z/OS components during normal production. During our testing, we used minimal values for these components. This does not apply if the traces are enabled temporarily for diagnostic reasons. The following command can be used to confirm the trace option status for each component: `DISPLAY TRACE`.

You can use this MVS command to dynamically disable the traces or turn the options to minimum (depending on the component): `TRACE,CT,OFF,COMP=x`
The CTIxxxyy PARMLIB members can be updated to harden any changes as well.

4. WLM Batch Initiator Model Training Resource Guidance

This section covers resource and tuning recommendations specific to model training.

4.1 Recommendations and Tuning Guidance

Below you will find some configuration recommendations based on observations done in a lab environment.

4.1.1 WLM Service Class Settings

Since the training is zIIP processor intensive, differentiating the service class and importance will allow WLM to service inferencing and higher priority zIIP work before the training work, which will run for a long time. It is important to take measures to ensure it will not interfere with other zIIP-eligible work on the system.

It is recommended to assign the training job to a different WLM service class from the MLz Core scoring service and high priority zIIP work. This service class should have a lower importance. You can do this by creating a WLM classification rule targeting the MLz Core Spark Driver and Executor address spaces (e.g. ALNSPKD* and ALNSPKX* respectively).

4.1.1.1 Honor Priority

Also, we recommend setting Honor Priority = No in the training job service class to ensure only zIIPs are being utilized for training, and the work will not spill over to CPs. In the example below, you can see how to set Honor Priority in the Modify a Service Class panel within the WLM ISPF Application.

Example:

```
                          Modify a Service Class
Command ===> ____________________________________________________________

Service Class Name . . . . . : PERFLO
Description  . . . . . . . . . Low  Priority work
Workload Name  . . . . . . . . PERFWKLD  (name or ?)
Base Resource Group  . . . . . ________  (name or ?)
Cpu Critical . . . . . . . . . NO        (YES or NO)
I/O Priority Group . . . . . . NORMAL    (NORMAL or HIGH)
Honor Priority . . . . . . . . NO        (DEFAULT or NO)
```

4.1.2 Spark resources

Even with proper WLM Service Class tuning, it is important to provide enough zIIP capacity so that other zIIP-eligible work can run efficiently while any training jobs are active. Training jobs run as Spark jobs and each Spark worker core can consume up to one full zIIP. By making sure there is plenty of zIIP capacity available for other zIIP-eligible work, you can further minimize the potential for zIIP-eligible work running on CPs.

You can control how many zIIPs are used by setting the SPARK_WORKER_CORE value. Unless there is no other zIIP-eligible work running on your system, it is recommended to set this value lower than the number of zIIPs on your system. However, the amount of resource you allow Spark to use will affect the duration of the training. You may choose to increase the number of zIIPs on the system to reduce training elapsed time without impacting other zIIP-eligible work. You can also plan training for times when it is less likely to impact other zIIP-eligible work.

4.1.2.1 SPARK_WORKER_CORES

A Spark job is broken down into multiple tasks that can be executed in parallel if you allocate enough processing resources to speed up the processing.

If you have multiple zIIPs available, you need to set the SPARK_WORKER_CORES parameter to be equal to the number of zIIPs you would like to use for the training.

Similarly, if you want to restrict the number of zIIPs that Spark can use for the training, you can use the SPARK_WORKER_CORES parameter to do so.

For example, with SPARK_WORKER_CORES=2, Spark will parallelize the training job on 2 zIIPs maximum.

The SPARK_WORKER_CORES parameter is found in the $IML_HOME/spark/conf/spark-env.sh configuration file.

Note: There is overhead associated with parallelization as you can see in our observation in section 4.2.1.2 Number of zIIPs. For that reason, we do not recommend using more than 4 zIIPs (4 SPARK_WORKER_CORES) for the training.

4.1.2.1 SMT-2

If you enabled SMT to take advantage of the extra logical zIIPs for the training, you should define 2 SPARK_WORKER_CORES per physical zIIP.

In our observation, the benefit of SMT is limited with the training workload, especially with 1 zIIP. It does seem to increase with the number of zIIPs utilized for the training.

4.1.3 Estimate Training Duration and zIIPs

Below are two formulae you can use to either estimate the training duration or estimate the number of zIIPs required.

Given the number of service classes ($\#service\ classes$) to train and the number of zIIPs ($\#zIIPs$) assigned to the training, you can estimate the duration (in minutes) of the training with the following formula:

$$\textbf{Training duration (minutes)} = \frac{\#service\ classes * 20}{\#zIIPs} * (1 + \frac{\#zIIPs - 1}{10}) + 2$$

"20" represents the average time in minutes it takes to train a single service class with a single zIIP.

The parallelization overhead previously mentioned and observed in section 4.2.1.2 Number of zIIPs is estimated via a factor that is function of the number of zIIPs.

The constant "2" represents the average time in minutes it takes for the Spark training job to initialize, perform tasks that are not dependent on the number of service classes or zIIPs, and finalize.

And vice versa, if you would like to estimate the number of zIIPs required to train a given number of service classes ($\#service\ classes$) in a given number of minutes ($Training\ duration\ (minutes)$), you can use this formula:

$$\#\boldsymbol{zIIPs} = \frac{18 \ * \ \#service\ classes}{Training\ duration\ (minutes) \ - \ (2 \ * \ \#service\ classes) \ - \ 2}$$

Note: The numbers given by the formulae are estimates for planning purposes only, not references. They are here to give an idea and help guiding when deciding how much resources to allocate for training and how long training may take.

The formulae were extrapolated from results obtained during lab measurements under lab conditions (system resources 100% dedicated to training, no other workload).

4.2 Observations

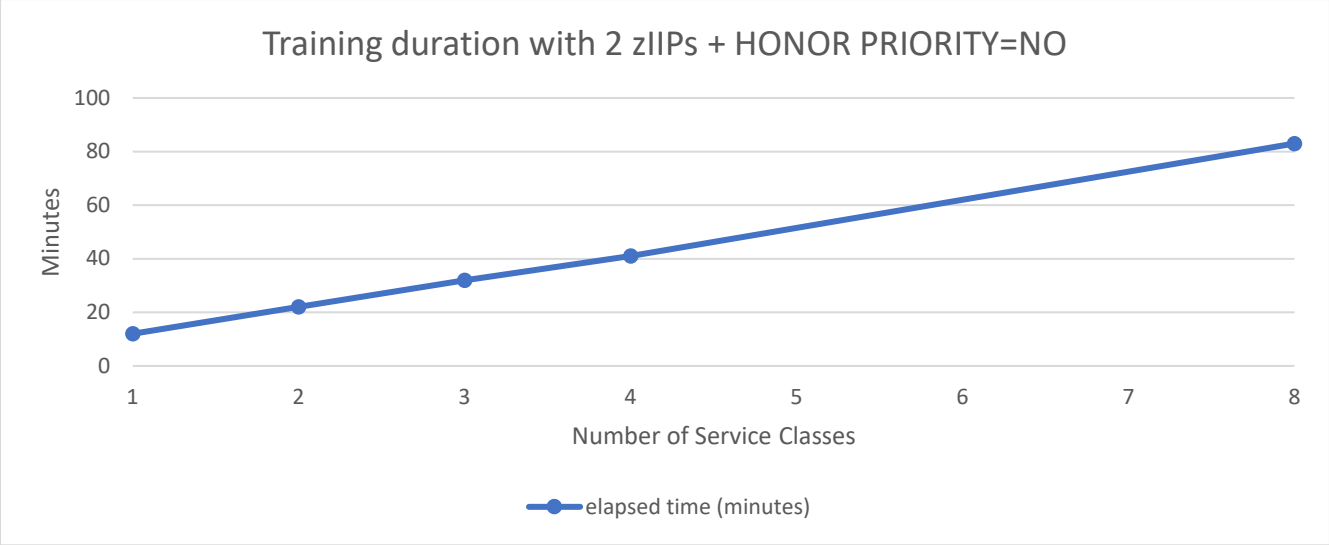
The recommendations from the previous section come from the following results that have been observed in a lab environment.

4.2.1 Training duration factors

4.2.1.1 Number of WLM Managed Service Classes

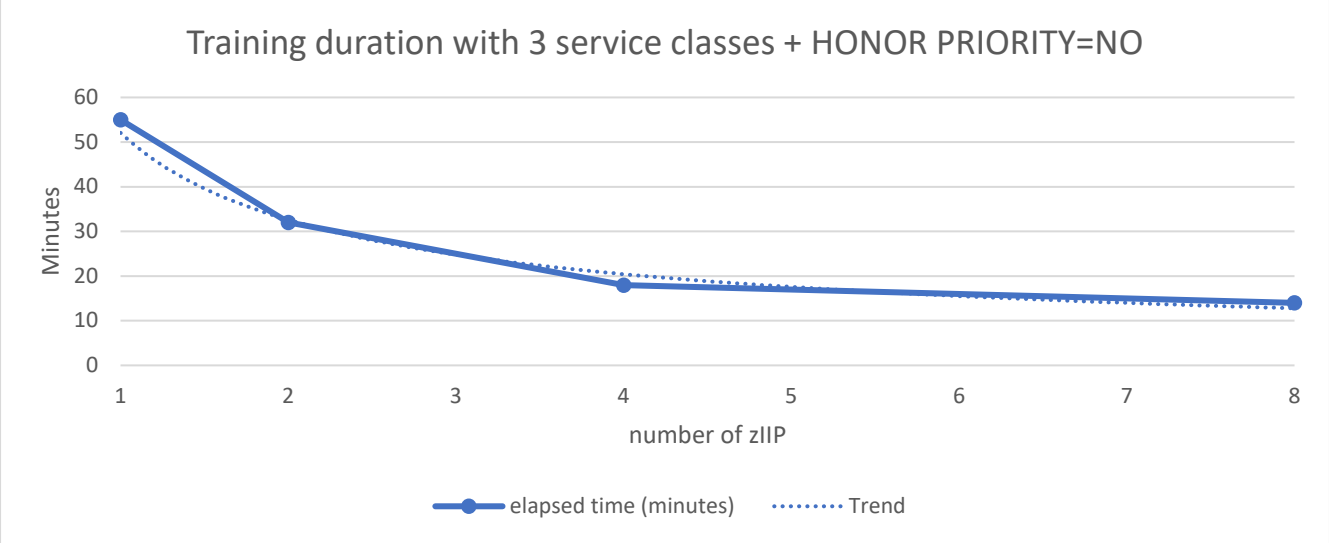
The more service classes you train, the longer the training will take.
Service classes are trained sequentially by the training job, so the training duration scales linearly with the number of service classes as you can see in the measurements below (taken on a system with 2 zIIPs):

# service classes	LPAR CP Util	LPAR zIIP Util	Training duration (minutes)
1	1%	87%	12
2	2%	87%	22
3	2%	87%	32
4	2%	87%	41
8	3%	84%	83



4.2.1.2 Number of zIIPs

Adding more zIIPs to increase Spark parallelization helps lower the training duration, but the benefit of adding zIIPs decreases as you add more zIIPs. You can see this effect in the graph below.



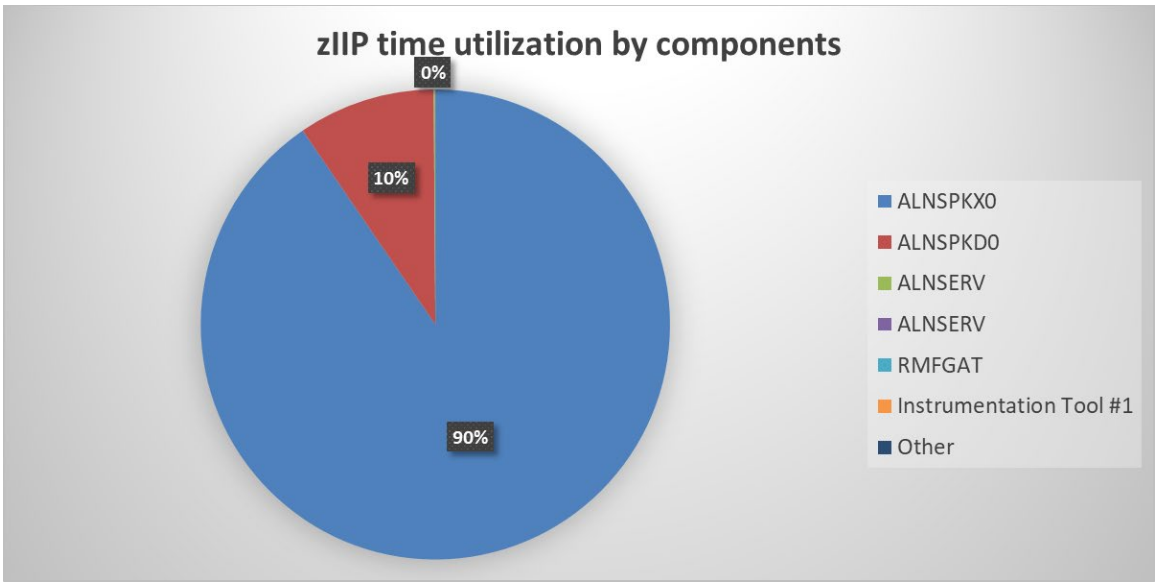
The overhead of the spark parallelization comes from the synchronization that Spark is doing in between each completed task. With more parallelism, more tasks are processed at the same time and more synchronization is required as each of them complete. In the next section, the detail CPU utilization graphs show the effect of the synchronization.

4.2.2 CPU Utilization

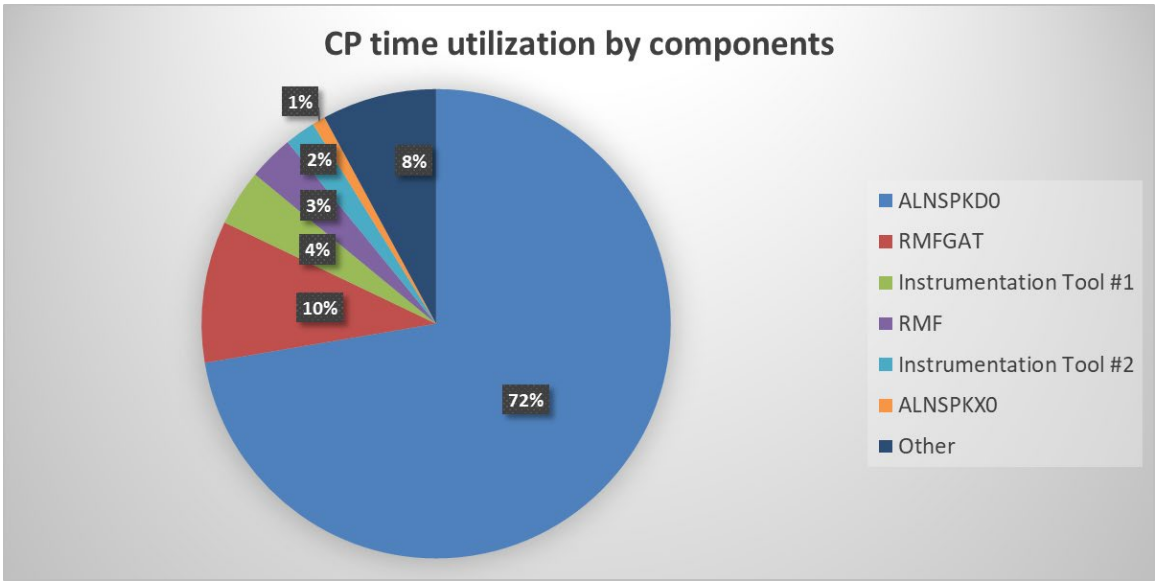
The higher the number of zIIPs (Spark parallelization) the lower the total zIIP utilization, hinting at some overhead due to synchronization within the parallelized job.

LPAR CPs	LPAR zIIPs	SPARK_WORKER_CORES	Service Classes	LPAR CP Utilization	LPAR zIIP Utilization	Elapsed Time (minutes)
1	1	1	3	2%	99%	55
1	2	2	3	2%	87%	32
1	4	4	3	2.5%	75%	19
1	8	8	3	4%	56%	14

Training is a CPU intensive workload which is largely zIIP eligible. The CP utilization is minimal during training.



Looking closer at how the time is spent on zIIP, the Spark Executor (ALNSPKX0) and Spark Driver (ALNSPKD0) are the 2 main components. The Spark Executor(s) oversee(s) the actual training and the Driver is managing and coordinating the executor(s).

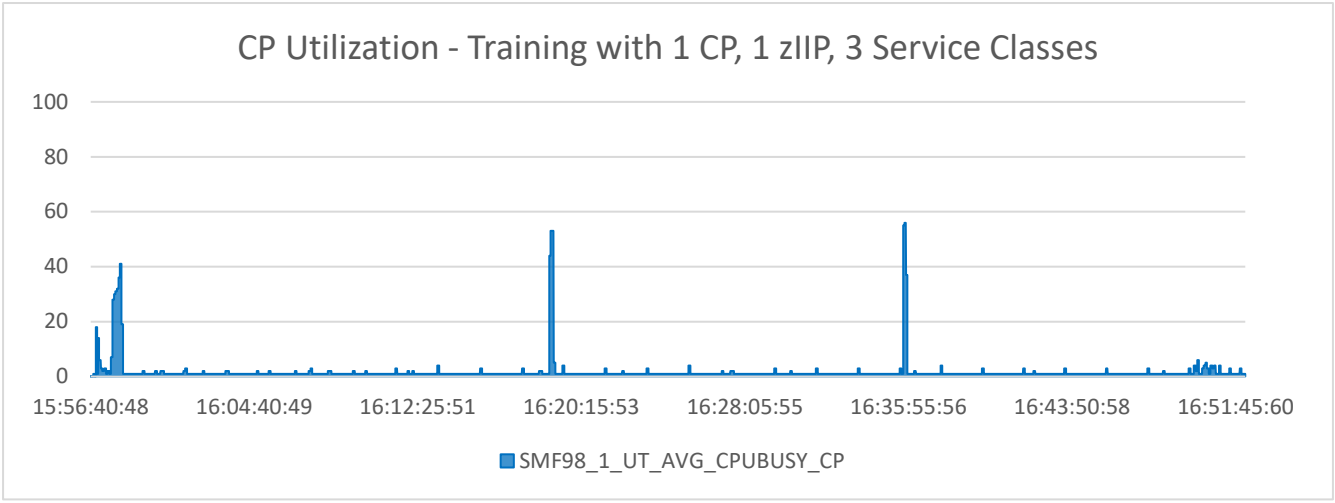


On the CP, the Spark Driver (ALNSPKD0) is the biggest time consumer. Most of the other components are related to monitoring and performance data capture (RMF, RMFGAT and other instrumentation tools we use). The Spark Executor (ALNSPKX0) has a very small share of the time spent on CP. Finally, 8% of the time on CP corresponds to other z/OS components and background activities.

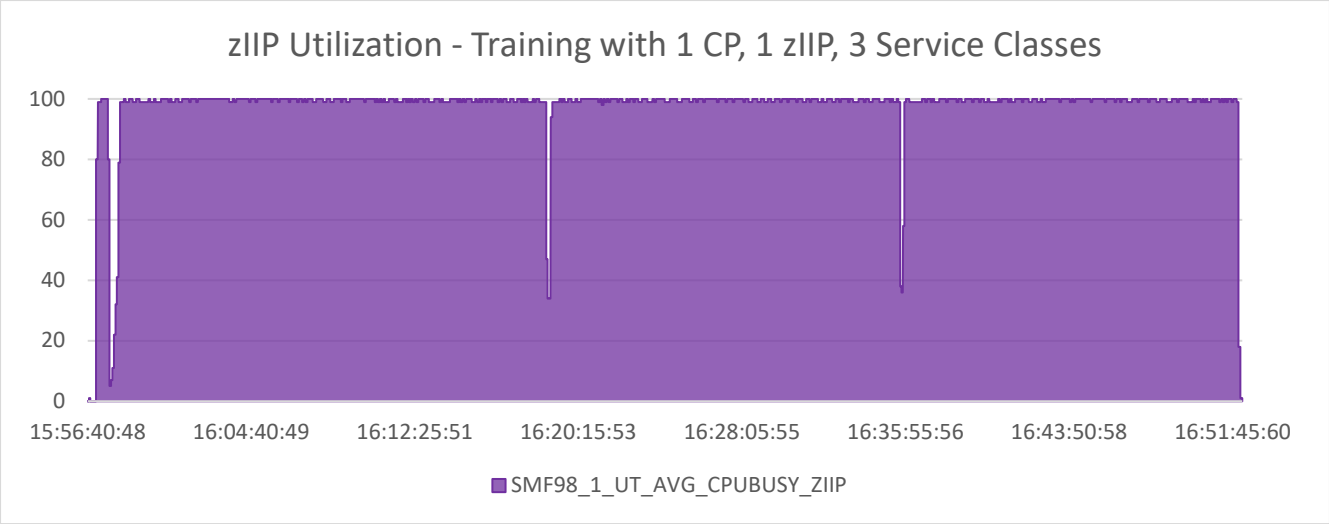
Overall, with the CP utilization being low (under 5% of a single CP in all the tests we did), we can conclude that the training workload is using minimal CP time.

The graphs below show the detailed CP and zIIP utilization during training with 1, 2, 4, and 8 zIIPs.

The two following graphs represent CP and zIIP utilization during training with 1 zIIP (and 1 SPARK_WORKER_CORES)

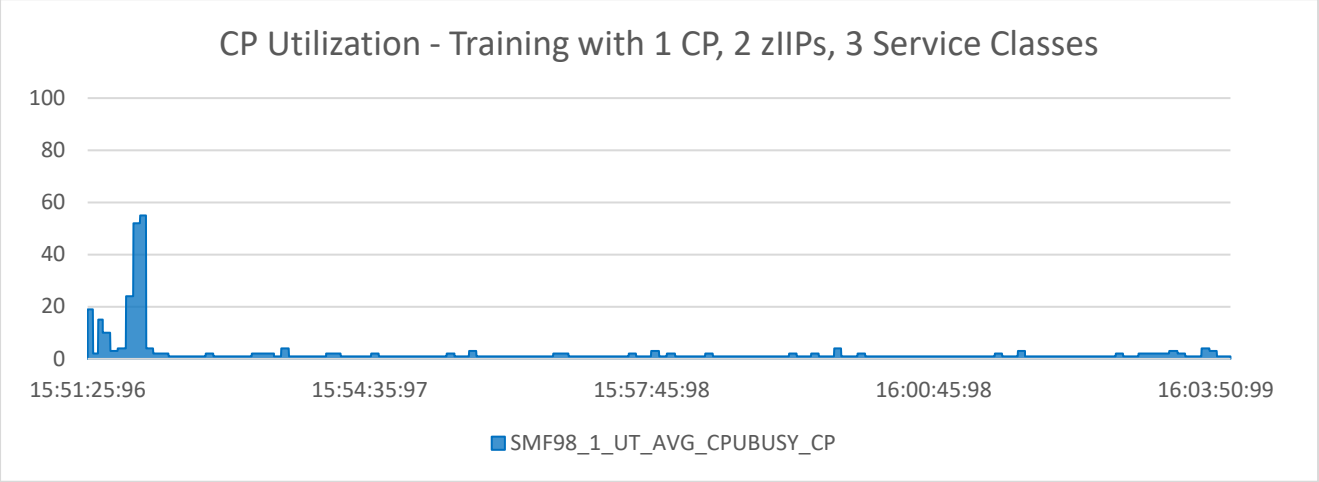


CP utilization is almost nonexistent, except for the small spike due to I/O when starting the training of the next service class.

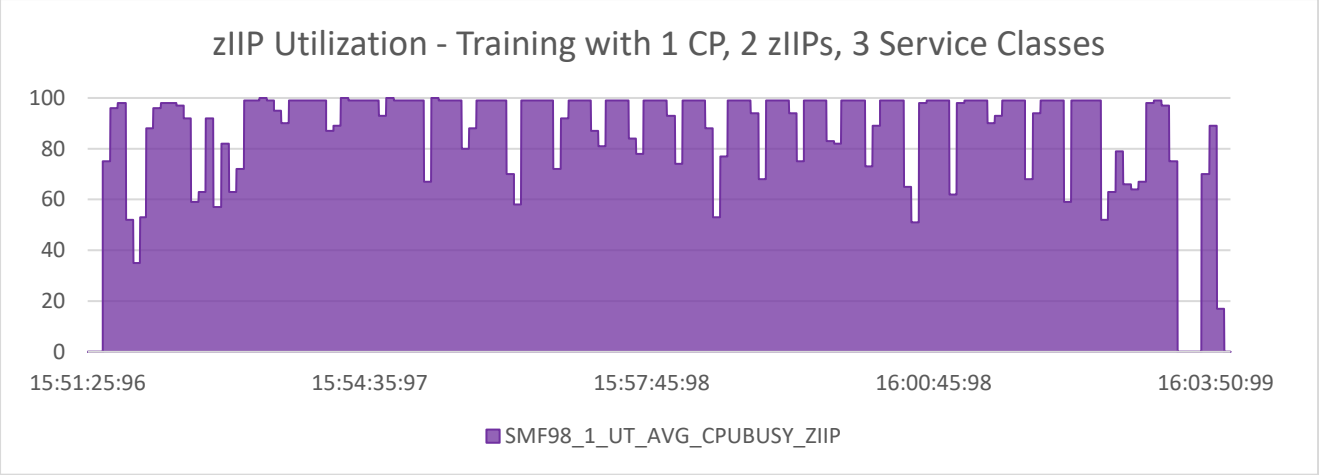


zIIP utilization is 100%, except for the small dips due to I/O when starting the training of the next service class.

The two following graphs represent CP and zIIP utilization during training with 2 zIIPs (and 2 SPARK_WORKER_CORES)

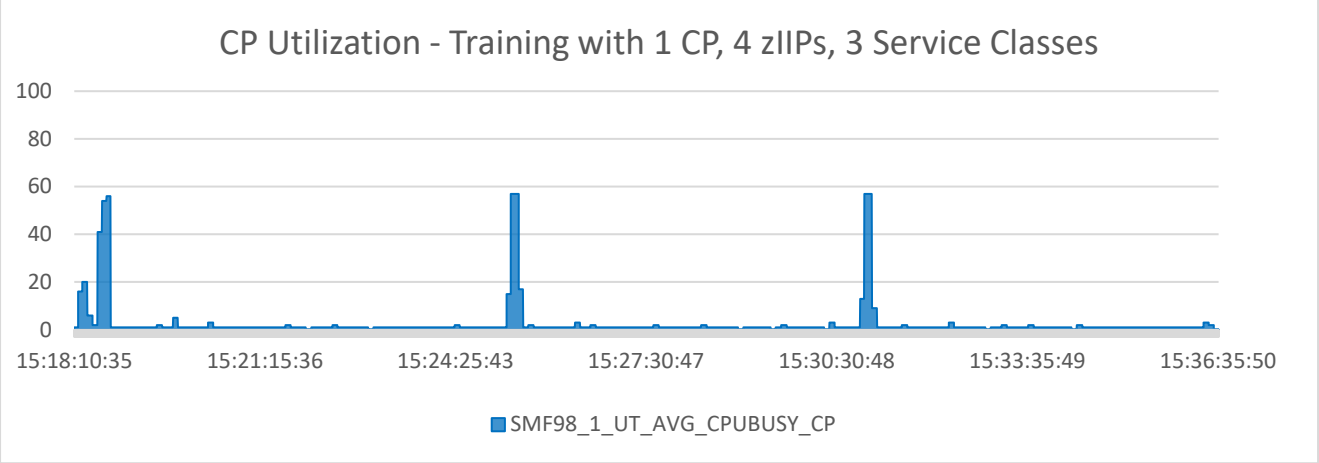


Like the 1 zIIP training, the CP utilization is very low, with a slightly bigger spike at the beginning of the training.

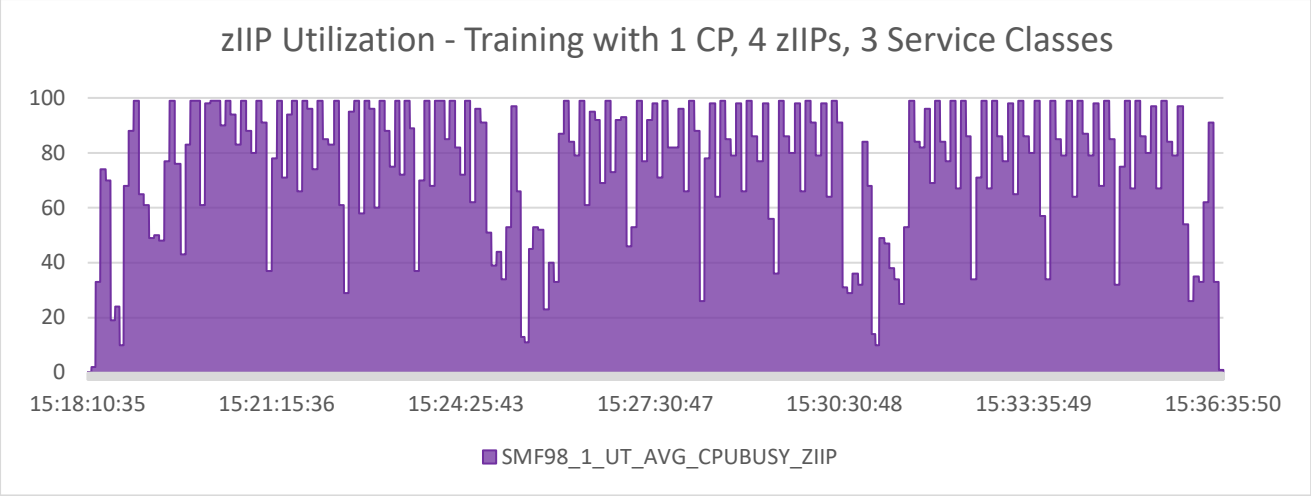


zIIP utilization is high, often at 100%, but there are small dips throughout the training duration due to synchronization between the parallel tasks executed on the 2 zIIPs.

The two following graphs represent CP and zIIP utilization during training with 4 zIIPs (and 4 SPARK_WORKER_CORES).

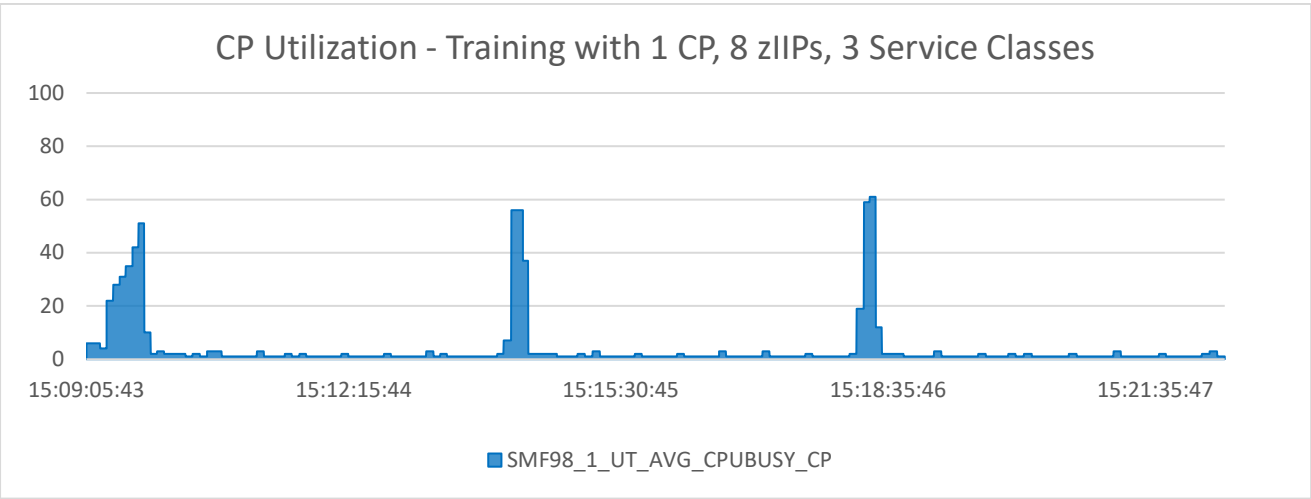


The overall CP utilization remains low, and the CP utilization pattern remains the same as we increase the number of zIIPs allocated to the training job.

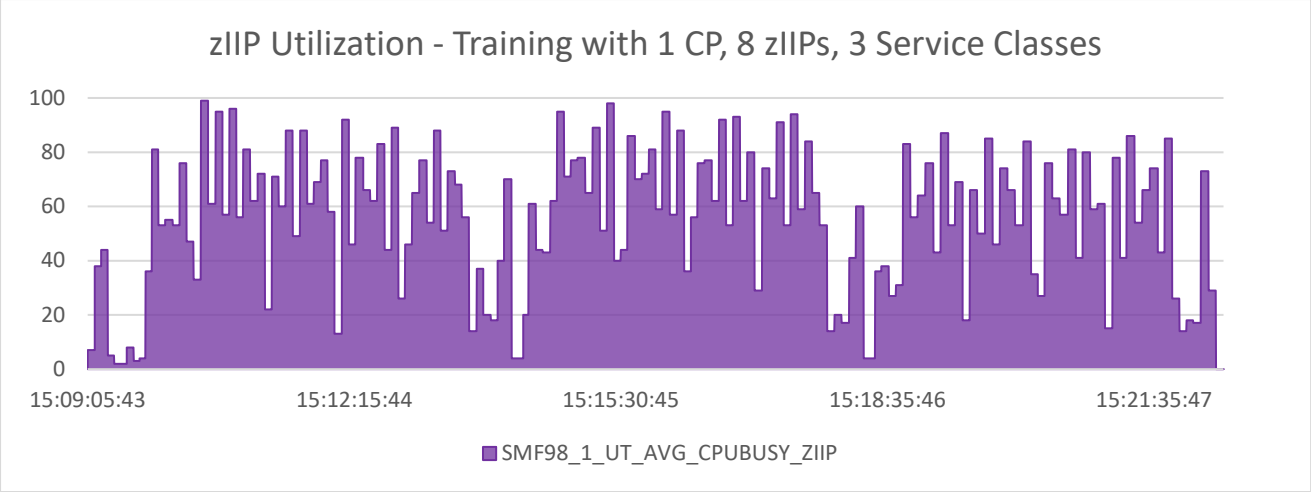


On the zIIP side, utilization is still high, but less often at 100%. The synchronization of the parallel tasks is more apparent here as it needs to synchronize more often with 4 tasks running parallel.

The two following graphs represent CP and zIIP utilization during training with 8 zIIPs (and 8 SPARK_WORKER_CORES).



The CP profile remains the same: low utilization with spikes due to I/O at the beginning of the training of each service class.



With 8 parallel tasks, the synchronization overhead is exacerbated: zIIP utilization is medium-high and spikes to 100% only for short periods of time due to the synchronization of the 8 parallel tasks.

4.2.3 Memory Utilization

When training the WLM Batch Initiator AI model, some memory is required by the Spark Driver and Executor which are both Java components.

With the default settings from the configuration workflow, the JVM for each of those components has a max heap size of 8 GB. Therefore, a total of 16 GB of memory can potentially be consumed during training by the Spark Driver and Executor.

During our testing, we found out that training the model with the lowest amount of memory allowed by the configuration workflow: 2 GB each for the Driver and the executor was sufficient. Therefore, we recommend setting those parameters to the minimum value allowed by the configuration workflow (2 GB at the time of writing this paper) to reduce the potential memory footprint of the training.

How to change the Java heap size for the Spark Driver and Executors:

1) Open $IML_HOME/configuration/deploy.cfg
2) Update the driver_memory setting to set the minimum and maximum heap size to the same desired value. If it does not already exist add the parameter to the end of the file.
3) Repeat step 3 for the executor_memory setting.
4) execute $IML_INSTALL_DIR/iml-services/create.sh

In addition to the memory required by the Spark Java components in charge of the training, the size of the buffer pools allocated to VSAM RLS matters to reduce the I/O done during training. See 3.1.1 IGDSMSxx PARMLIB Settings for tuning guidance.

5. Conclusion

For AI-powered WLM batch initiator management and the z/OS 3.1 AI framework, we have taken several key resource and tuning aspects into consideration including memory, zIIP processors, and VSAM RLS tuning. Based on our test outcomes, findings, recommendations, and tuning guidance, we would like help you leverage the z/OS 3.1 AI capabilities while reducing the potential impact on other workloads and the system.

As AI models tend to have a larger memory footprint compared to traditional z/OS workloads, it is important to make sure that you have the necessary memory available. Additional memory can also be used to increase buffer sizes, allowing you to further optimize performance.

In addition, the offering is highly zIIP eligible, and the AI model training can consume considerable zIIP CPU, depending on how you configure Spark. It is important to be prepared from a planning and WLM policy perspective for model training to avoid impacting parallel zIIP-eligible work.

By properly tuning VSAM RLS to account for this offering as well as future use cases, data can be sourced from memory more often, helping optimize EzNoSQL and overall VSAM RLS performance.

Further, monitoring system performance and being equipped to adjust based on the data is equally as important as sizing resources correctly from the start.

As this is the first foray from IBM of infusing AI into the z/OS operating system, you might expect that the space will grow and change rapidly, as new use cases and enhancements are brought onboard. Stay tuned!

IBM Corporation
New Orchard Road
Armonk, NY 10504

Performance and resource consumption information is based on measurements and projections using simulated workloads and data, executed in a controlled environment. The actual results that any user will experience will vary depending upon considerations such as the number of WLM managed service classes, historical data, the processor, memory, network, coupling facility, and I/O configurations, LPAR configuration, and workloads present. Therefore, no assurance can be given that users will experience the same results stated here. The information is subject to change, and this document may be updated.