International Journal on Science and Technology (IJSAT)
E-ISSN: 2229-7677 ● Website: www.ijsat.org ● Email: editor@ijsat.org

Technical Insights into the zOS file system and datasets

Chandra mouli Yalamanchili

chandu85@gmail.com

Abstract

The z/OS operating system offers a strong and well- organized dataset management System that enables the efficient storage, retrieval, and processing of data through high-volume and mission - critical workloads by large corporations.

This paper delves into the intricacy of the z/OS operating system's file management, more so its unique but most sophisticated dataset structures and UNIX System Services (USS) file system. This paper explores various dataset organizations, record structures, and access methods native to z/OS. This paper also talks about how the traditional z/OS datasets integrate with the hierarchical file system of USS, with a focus on coexistence and interoperation between the two environments.

This paper aims to comprehensively understand the z/OS file systems and datasets by comparing similar technologies outside of zOS and using coding examples.

Keywords: z/OS datasets; UNIX System Services file system; VSAM; HFS; zFS; data management; mainframe storage

Introduction

IBM's z/OS is a robust mainframe computer operating system with unparalleled scalability, security, and reliability. At the center of its architecture is a sophisticated file management system that is quite distinct from file systems found in other operating systems.

z/OS employs datasets as its primary data storage entities, each carefully designed to fulfill various data processing requirements. Alongside UNIX System Services (USS), z/OS has aUNIX-like hierarchical file system supporting UNIX-based applications.

This paper provides an in-depth exploration of these file systems, explaining their structures, functionalities, and synergy.

z/OS Storage Architecture

z/OS uses Direct Access Storage Devices (DASD) to manage storage, which provides high-performance disk storage optimized for enterprises' mission-critical workloads. z/OS uses volume-based allocation and system-managed storage to provide high efficiency [12].

z/OS also supports tape storage, but DASD remains the primary medium for file systems and datasets [9,10].

DASD and Volume Management

- DASD (Direct Access Storage Device): A disk-based storage system that organizes data into cylinders and tracks for efficient access. [9, 10]
- Volumes: Each DASD is divided into volumes, which act as logical storage units. Each volume is uniquely identified by a Volume Serial Number (VOLSER). [9]
- Extent Allocation: When a dataset is created, it is allocated space on a volume in extents, which are contiguous or non-contiguous blocks of storage. [10]

Storage Management in z/OS

- System Managed Storage (SMS): A policy-based system that simplifies many dataset operations by automating storage management tasks. SMS helps administrators effectively manage DASD storage resources based on predefined rules. Below are a few of the most significant functions of SMS [19]:
    - Automated dataset placement: SMS can determine where and how datasets are placed without storage attribute specification based on preconfigured policies. [9][10]
    - Storage Class assignment: Defines performance attributes (e.g., caching, types of devices) for datasets.[9][10]
    - Management Class: Manages dataset's retention policies, backup frequency, and migration rules.[9][10]
    - Data Class: Manages records' size, format, and allocation behavior.[9][10]
    - Storage Group: SMS groups the volumes for efficient storage management.[9][10]
- Catalogs:
    - Master Catalog: The primary catalog that keeps track of all datasets and their locations. [1][10]
    - User Catalogs: Additional catalogs that help manage dataset locations for different applications and users. [1]
- VSAM and Non-VSAM Storage: VSAM datasets have their own indexing and management, while non-VSAM datasets are stored directly on DASD volumes [5][9].

z/OS Datasets: Structure and Organization

In z/OS, a dataset is similar to a file in other operating systems, serving as a repository for storing and managing records. Datasets are characterized by specific attributes that define their organization, record format, and access methods.

Dataset Organization (DSORG)

- Physical Sequential (PS): Datasets with a sequential organization where records are stored one after another. They are ideal for batch processing and are analogous to flat files in other systems.[1][5]
- Partitioned Organization (PO): Also known as Partitioned Data Sets (PDS), these datasets contain members, each functioning like a separate sequential dataset. This structure will help group related datasets, such as program libraries.[1][5]
- Virtual Storage Access Method (VSAM): A sophisticated and advanced access method that supports different types of data organizations as below:
    - Key-Sequenced Data Set (KSDS): Records are organized based on a key field, facilitating efficient retrieval. [5]
    - Entry-Sequenced Data Set (ESDS): Records are stored sequentially as they are entered.[5]
    - Relative Record Data Set (RRDS): Records are accessed based on their relative position within the dataset.[5]
    - Linear Data Set (LDS): Consists of a sequence of pages with no intrinsic record structure, often used for memory-mapped files. [5]

Record Formats (RECFM)

- Fixed (F): Each record has a fixed length, simplifying processing but potentially leading to space inefficiencies if record content varies significantly. [5]
- Variable (V): Records can have differing lengths, each preceded by a Record Descriptor Word (RDW) indicating its size. This format is space-efficient for datasets with records of varying lengths, but it will add more complexity to processing. [5]
- Undefined (U): Records have no predefined structure, providing more flexibility, but this format requires more complicated and custom processing logic. [5]

Access Methods

- Queued Sequential Access Method (QSAM): Optimized for sequential processing of datasets, commonly used for reading or writing large volumes of data in order. [5]
- Basic Direct Access Method (BDAM): Allows direct access to records based on their physical location on the storage device, suitable for applications requiring rapid, non-sequential data retrieval. [5]
- Virtual Storage Access Method (VSAM): Provides advanced data management capabilities, supporting complex dataset structures like KSDS, ESDS, and RRDS. [5]

Comparison of Dataset Organizations and Access Methods

Feature	Physical Sequential (PS)	Partitioned Organization (PO/PDS)	VSAM - KSDS	VSAM - ESDS	VSAM - RRDS	VSAM - LDS
Structure	Flat file, sequential records	Directory-like structure with members	Indexed file with key lookup	Sequential records only	Records accessed by relative number	Linear, page-based structure
Use Case	Batch processing, logs, sequential data	Libraries, program storage	Indexed access for high-performance applications	Append-only logs, sequential access	High-speed relative access	Database and system storage
Supports Random Access	No	Indirectly via members	Yes	No	Yes	Yes
Supports Sequential Access	Yes	Yes	Yes	Yes	Yes	No

Table 1: Comparison of different dataset organizations [1][5][9]

Access Method	QSAM	BDAM	VSAM (KSDS, ESDS, RRDS)
Primary Use	Sequential file processing	Direct access to physical records	Indexed and sequential data access
Supports Buffering	Yes	No	Yes
Supports Sequential Access	Yes	No	Yes
Supports Random Access	No	Yes	Yes
Commonly Used With	PS datasets	BDAM datasets	VSAM datasets

Table 2: Comparison of different access methods[1][5][9]

Deep Dive into VSAM

The Virtual Storage Access Method (VSAM) is a highly efficient data access method in z/OS that handles structured data. It is widely used for high-performance record access in large enterprise applications such as banking and transaction processing. [5]

Unlike traditional sequential datasets, VSAM provides indexed and direct access to records, which improves performance and scalability. It also has advanced storage management techniques, including Control Intervals (CI), Control Areas (CA), and buffering algorithms to improve data access.[5]

The following sections describe significant enhancements that improve the features and availability of VSAM in enterprise environments.

Key Features of VSAM

Control Interval (CI) and Control Area (CA)

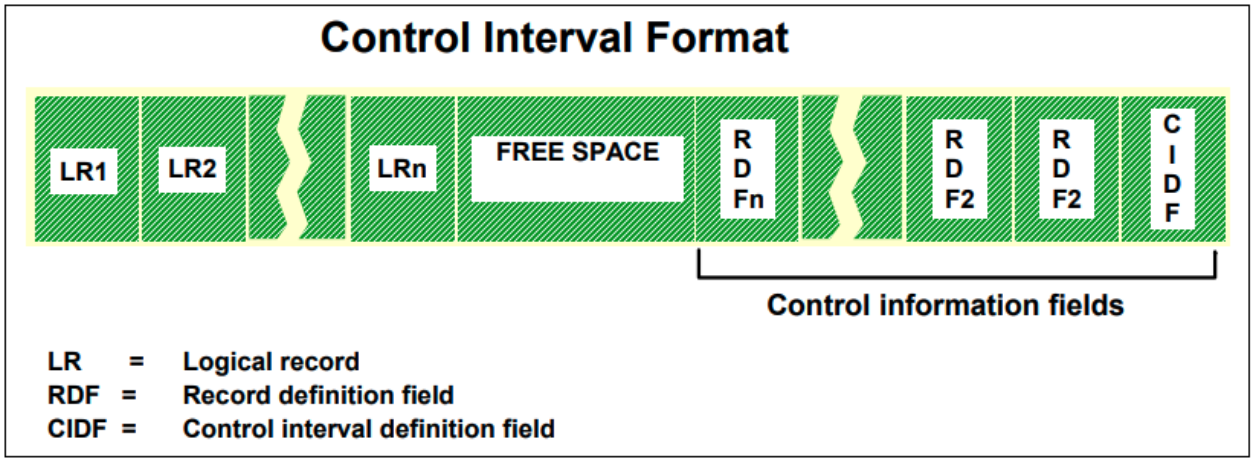

Figure 1: Illustrating the general format of a control interval. [5]

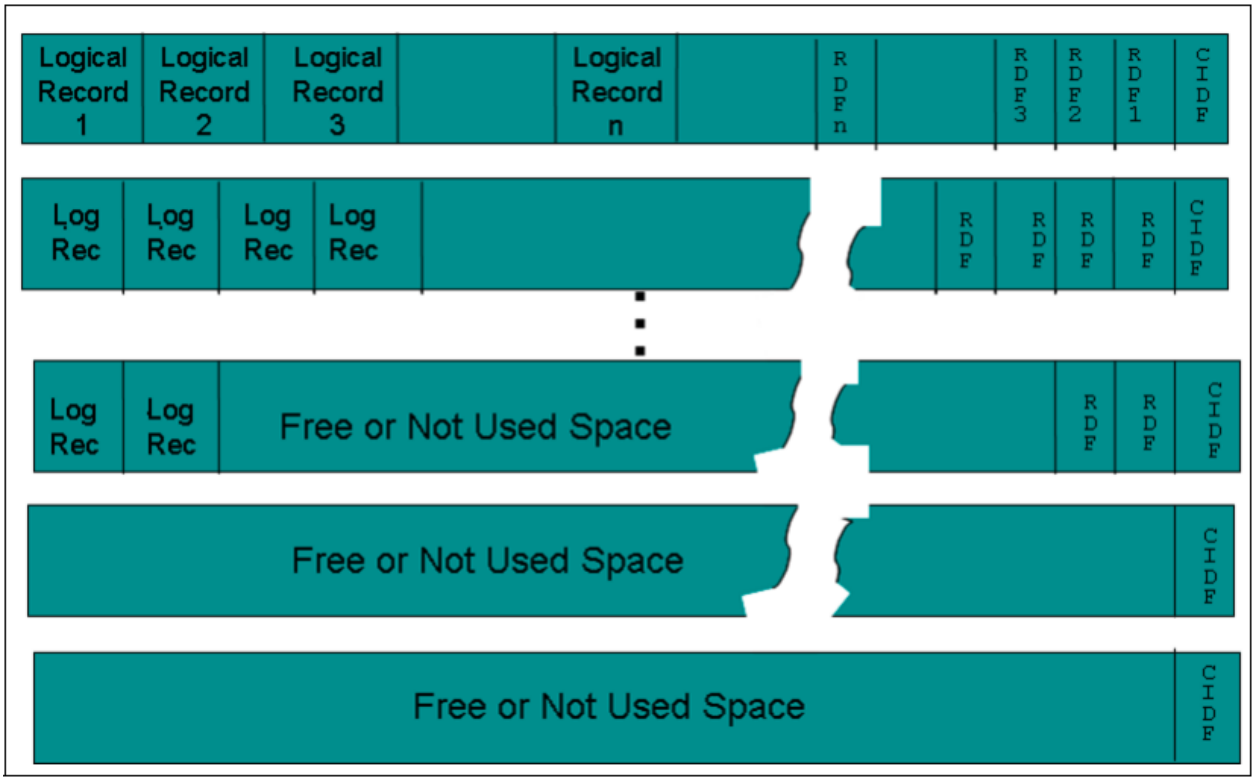

Figure 2: General illustration of CA with several CIs

- **Control Interval (CI):** The smallest data transfer unit in VSAM, similar to a block in traditional file systems. CI is the continuous area of the DASD volume track that VSAM uses to store the data records, and the control information related to that record. [5]
    - A CIDF is a 4-byte field that contains the information about the amount and location of free space available within CI. There will be on CIDF per CI. [5]
    - An RDF is a 3-byte field that describes the length of records. For variable size records, there will be one RDF for each logical record. For fixed length records there will be two RDFs, one with the length and other with how many with that length. There will be several RIDFs per CI depending on number of logical records and their type. [5]
- **Control Area (CA):** A collection of Control Intervals allocated together for efficient space management. [5]
- These structures help VSAM datasets achieve high access speeds and minimize data fragmentation. [5]

System-Managed Storage for VSAM (SMSVSAM)

- SMSVSAM automates VSAM data set storage management. [5]
- It allows for datasets to be dynamically assigned without human intervention. [5]
- Enhances performance by optimizing space utilization and system calibration. [5]

Coupling Facility (CF) for VSAM

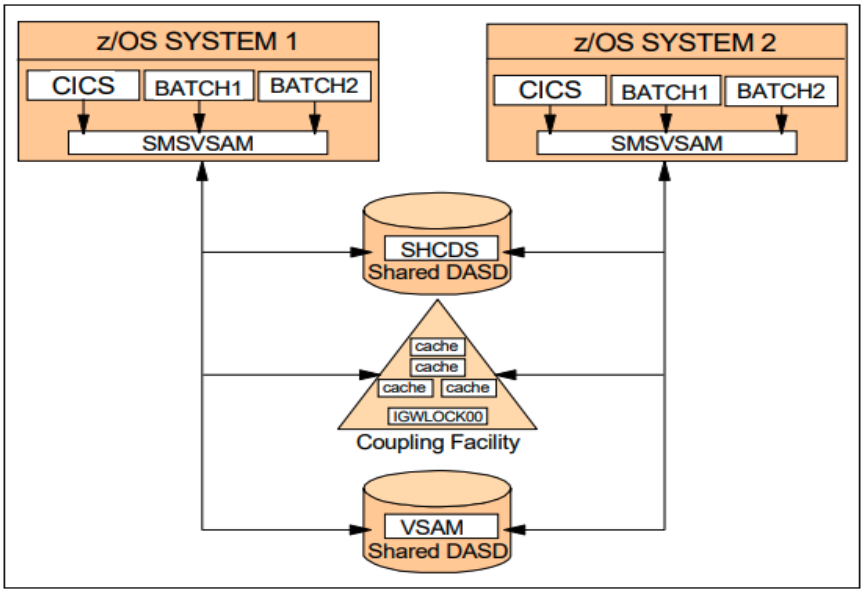

Figure 3: RLS in action, illustrating Coupling Facility. [5]

- The Coupling Facility enables VSAM datasets to be accessed by multiple Logical Partitions (LPARs) in a Sysplex setup, supports real-time data performance, and minimizes the contention on the data [11].
- Above picture reflects a simplified view of VSAM data set being shared between different address spaces across a different Sysplex. Each z/OS system in the Sysplex has its own SMSVSAM address space to coordinate the sharing. Sharing Control Data Set (SHCDS) contains critical information used by various SMSVSAM address spaces for RLS data set access.
- Coupling facility offer record level locking capabilities as well as caching capabilities to enable efficient sharing of the VSAM file across different systems. [5]

Record Level Sharing (RLS)

- RLS enables record-level locking instead of dataset-level locking. [5]
- Allows multiple applications to access VSAM datasets concurrently. [5]
- Minimize bottlenecks and improve transaction throughput. [5]

Buffering and Caching

- VSAM datasets use **Local Shared Resources (LSR)** and **Non-Shared Resources (NSR)** to optimize the data buffering/caching.
- LSR is used for high-speed indexed data access, while NSR is suited for batch processing.
- Buffering mechanisms help to reduce the overhead of disk I/O. [5]

VSAM Cluster Components

A **VSAM cluster** consists of the following [5]:

- **Data Component:** Stores actual data records.
- **Index Component:** Maintains key-based indexing for quick searches and data access based on partial or full keys.
- **Catalog Entry:** Manages metadata for dataset organization.

Code Examples

Defining a VSAM KSDS dataset

IBM's IDCAMS utility can be used to allocate VSAM dataset, below is an example of how a KSDS (Key-Sequenced Dataset) can be allocated.

```
//DEFINEKS JOB (ACCT),'DEFINE KSDS',CLASS=A,MSGCLASS=A
//STEP1    EXEC PGM=IDCAMS
//SYSPRINT DD  SYSOUT=A
//SYSIN    DD  *
  DEFINE CLUSTER (NAME(USERID.VSAM.KSDS) -
    INDEXED -
KEYS(10 0) -
RECORDSIZE(100 1000) -
CISZ(4096) -
    VOLUMES(DEV001) -
SHR(2,3) )
  DATA (NAME(USERID.VSAM.KSDS.DATA))
  INDEX (NAME(USERID.VSAM.KSDS.INDEX))
/*
```

Accessing a VSAM KSDS file in COBOL

Below is an example of how a KSDS (Key-Sequenced Dataset) can be access in COBOL program.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. VSAM-RW.

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT VSAM-FILE ASSIGN TO 'USERID.VSAM.KSDS'
    ORGANIZATION IS INDEXED
    ACCESS MODE IS SEQUENTIAL
    RECORD KEY IS VSAM-KEY
    FILE STATUS IS WS-FILE-STATUS.

DATA DIVISION.
FILE SECTION.
FD VSAM-FILE.
01 VSAM-RECORD.
   05 VSAM-KEY  PIC X(10).
   05 VSAM-DATA PIC X(90).

WORKING-STORAGE SECTION.
01 WS-FILE-STATUS PIC X(2).
01 NEW-RECORD.
   05 NEW-KEY  PIC X(10) VALUE 'NEWKEY001'.
```

```
   05 NEW-DATA PIC X(90) VALUE 'This is a new record.'.

PROCEDURE DIVISION.
   OPEN I-O VSAM-FILE.

   DISPLAY 'Writing a new record...'.
   WRITE VSAM-RECORD FROM NEW-RECORD
   INVALID KEY DISPLAY 'Error: Write failed.'
   NOT INVALID KEY DISPLAY 'Record successfully written.'.

   DISPLAY 'Reading records...'.
   PERFORM UNTIL WS-FILE-STATUS = '10'
      READ VSAM-FILE NEXT RECORD
         AT END MOVE '10' TO WS-FILE-STATUS
         NOT AT END DISPLAY 'READ: ' VSAM-RECORD
   END-PERFORM.

   CLOSE VSAM-FILE.
   STOP RUN.
```

VSAM Summary

VSAM access method provide a powerful, flexible, and high-performance method for managing and accessing structured data within z/OS. With features like SMSVSAM, Coupling Facility, and Record-Level Sharing, it ensures scalability, concurrency, and efficient data retrieval for enterprise applications.

UNIX System Services (USS) Integration

With the introduction of USS, z/OS extended its capabilities to support a hierarchical file system similar to UNIX and Linux-based operating systems. It allows z/OS to natively run UNIX-based applications, which bridges the gap between traditional mainframe datasets and modern file system designs.

USS File Systems

- **HFS:** The first implementation of a UNIX-like file system in z/OS, supporting hierarchical directory structures and POSIX-compliant file operations. However, HFS was limited in performance and scalability.[2]
- **zFS:** Being a next-generation file system engineered to overcome HFS limitations, zFS offers improved performance, scalability, and reliability. It features dynamic growth, concurrent access, and enhanced recovery mechanisms.[2]

Mounting File Systems

In z/OS, file systems like HFS or zFS are mounted to put them within the hierarchical directory structure. Each file system is typically associated with a specific dataset, and mounting makes its contents available within the USS environment [2].

Accessing Datasets from USS

USS provides mechanisms to access traditional z/OS datasets, facilitating interoperability between the two file systems. For instance, UNIX commands can interact with datasets using special path notations.

Example: To list the contents of a PDS member using the cat command:

```
cat "//'USERID.DATASET(MEMBER)'"
```

Java code example to read a VSAM KSDS file

The following Java program demonstrates how to read a VSAM KSDS file in USS environment:

```
import java.io.*;
import java.nio.file.*;
import java.nio.charset.StandardCharsets;

public class VSAMReader {
    public static void main(String[] args) {
        String vsamFilePath = "//USERID.VSAM.KSDS"; // Replace with actual VSAM dataset name

        try (BufferedReaderbr = Files.newBufferedReader(Paths.get(vsamFilePath),
StandardCharsets.UTF_8)) {
            String line;
            while ((line = br.readLine()) != null) {
System.out.println(line);
            }
        } catch (IOException e) {
System.err.println("Error reading VSAM file: " + e.getMessage());
e.printStackTrace();
        }
    }
}
```

This Java code reads the VSAM KSDS dataset using USS file handling capabilities and prints each record to the console.

Java Code Example Using JZOS SDK to Read a VSAM KSDS File

IBM provides JZOS SDK, a set of Java APIs designed specifically for accessing z/OS datasets, including VSAM files, more efficiently. JZOS integrates with the z/OS system and provides native access to datasets, making it ideal for large-scale enterprise processing.

The following Java program reads & writes into a VSAM KSDS file using JZOS RecordReader & JZOS RecordWriter:

```
import com.ibm.jzos.RecordWriter;
import com.ibm.jzos.TextRecordReader;
import com.ibm.jzos.TextRecordWriter;
import java.io.IOException;

public class JZOSVSAMReadWrite {
    public static void main(String[] args) {
        String datasetName = "//'USERID.VSAM.KSDS'"; // Replace with actual VSAM dataset name

        // Write a record to VSAM
writeToVSAM(datasetName, "NEWKEY003", "This is a new VSAM record using JZOS.");

        // Read all records from VSAM
readFromVSAM(datasetName);
    }

    public static void writeToVSAM(String datasetName, String key, String data) {
        try (RecordWriter writer = new TextRecordWriter(datasetName)) {
            String fullRecord = String.format("%-10s%-90s", key, data); // Ensure key and data fit the
expected format
writer.write(fullRecord);
System.out.println("Successfully written to VSAM: " + fullRecord);
        } catch (IOException e) {
System.err.println("Error writing to VSAM file: " + e.getMessage());
e.printStackTrace();
        }
    }

    public static void readFromVSAM(String datasetName) {
        try (TextRecordReader reader = new TextRecordReader(datasetName)) {
            String record;
System.out.println("Reading records from VSAM...");
            while ((record = reader.read()) != null) {
System.out.println("READ: " + record);
            }
```

```
        } catch (IOException e) {
System.err.println("Error reading VSAM file: " + e.getMessage());
e.printStackTrace();
        }
    }
}
```

Why Use JZOS SDK?

- Optimized for z/OS - JZOS provides native integration with mainframe datasets. [7]
- Efficient Dataset Processing -It supports large-scale batch jobs more effectively than standard Java file I/O. [7]
- Better Performance for VSAM - It avoids issues when using standard Java NIO with VSAM datasets. [7]
- JZOS provides the ability to read VSAM datasets by key, allowing direct retrieval of records using indexed access, which standard Java NIO does not support. This capability makes JZOS essential for applications requiring efficient, direct access to specific records in a VSAM KSDS dataset.[7]

JZOS is recommended for production-grade applications where performance, scalability, and native integration with z/OS datasets are critical [7].

Comparisons with Similar Technologies

Below are a few distinctions when comparing z/OS datasets and file systems to those in other operating systems:

- **Windows NTFS and UNIX ext4:**
  - Both are hierarchical file systems supporting directories and files with various attributes [1].
  - In contrast, z/OS datasets are not inherently hierarchical and are managed through catalogs [1,5].
- Database Management Systems (DBMS):
  - Modern DBMSs handle structured data with complex querying capabilities [5].
  - z/OS datasets, especially VSAM datasets, provide foundational data storage mechanisms that DBMSs can utilize [5].
- **Cloud-Based Storage Models (AWS S3, Azure Blob, Google Cloud Storage):**
  - Cloud storage solutions provide scalable, distributed storage accessible over the internet [1].

- Unlike z/OS datasets, which are tightly integrated with transaction processing and structured storage management, cloud storage focuses on object-based storage that scales dynamically with demand [1,9].
- Cloud models offer high availability, redundancy, and accessibility but lack the deep integration with mainframe transaction processing that z/OS datasets provide [1,5].

Conclusion

z/OS's file management system is a cornerstone of its architecture, offering diverse dataset structures and access methods tailored for high-performance computing environments. The Virtual Storage Access Method (VSAM) provides structured data management with support for indexed, sequential, and direct access methods, while enhancements like SMSVSAM, Coupling Facility, and Record Level Sharing (RLS) improve scalability and availability. The integration of UNIX System Services (USS) further enhances flexibility by enabling hierarchical file system support and interoperability with UNIX-based applications.

Compared to traditional hierarchical storage models like NTFS and ext4, z/OS datasets rely on catalog-based organization, while cloud-based storage models like AWS S3 offer scalability but lack deep integration with transactional processing. Future research should focus on optimizing dataset management for modern workloads, enhancing file system security, and integrating cloud storage capabilities with z/OS.

To conclude, with its unmatched reliability and scalability, z/OS will continue to be a core foundation for enterprise data management in the evolving digital landscape.

References

[1] IBM Corporation, "z/OS DFSMS: Using Data Sets", IBM Redbooks, 2020.

[2] IBM Corporation, "z/OS UNIX System Services Overview", IBM Redbooks, 2020.

[3] M. Ebbers, J. Kettner, and W. O'Brien, "Introduction to the New Mainframe: z/OS Basics", IBM Redbooks, 2019.

[4] IBM Corporation, "z/OS File Systems: Managing and Implementing zFS", IBM Redbooks, 2019.

[5] IBM Corporation, "VSAM Demystified", IBM Redbooks, 2018.

[6] J. Varady, "Advanced VSAM: Techniques for High-Performance Processing", McGraw-Hill, 2017.

[7] IBM Corporation, "JZOS Guide: Java on z/OS", IBM Documentation, 2019.

[8] R. Barker, "Mainframe Storage Management: Best Practices and Strategies", Wiley, 2016.

[9] IBM Corporation, "DFSMS: Managing Storage on z/OS", IBM Redbooks, 2018.

[10] IBM Corporation, "Introduction to DASD Management on z/OS", IBM Redbooks, 2019.

[11] C. Yalamanchili, "IBM Mainframe & z/OS: Advanced Insights from A Programmer's Perspective", International Journal for Multidisciplinary Research (IJFMR), Vol. 2, Issue 3, May-June 2020, doi: 10.36948/ijfmr.2020.v02i03.22604.

[12] C. Yalamanchili, "Technical Insights into the Implementation of z/OS Memory and Address Spaces", International Journal on Science and Technology (IJSAT), Vol. 11, Issue 4, October-December 2020, doi: 10.5281/zenodo.14288200.