Vol.10No.12(2019),92-109
DOI: https://doi.org/10.61841/turcomat.v10i12.14954

Building Scalable Batch Processing Systems for Financial Transactions Using Mainframe

Akash Gill

Abstract
Using mainframes to develop highly scalable batch-processing systems for financial transactions is mandatory for banking solutions with a high transaction rate. In this article, the author focuses on the role of Mainframes in making financial systems effective, scalable, and robust. One of the technologies widely used in financial institutions is batch processing, which helps institutions to complete large volumes of data processing at low costs, bearing high costs on both hardware and human resources to complete the transactions. The article is more of a technical nature, exploring the principles of building and deploying large and extendible batch processing environments with an emphasis on using COBOL for program development, JCL for scheduling, and DB2 for integration. In this regard, the principal approaches include job prioritization, load balancing, and error recovery measures, all discussed to improve performance and reduce downtime. The result of deploying such systems consists of minimizing the time it takes to complete the various transactions and adapting to the expanding needs at the corporate level. Mainframe technology has user-friendly functions and capabilities that are suitable for today's financial institutions because these institutions require systems that provide services to customers effectively and safely with extensive scalability. Therefore, it is essential to point out from this article that Mainframes remain critical infrastructure in the move to digital financial services.

Keywords: Scalable Batch Processing, Mainframe Systems, Financial Transactions, COBOL Programming, JCL (Job Control Language), DB2 Database Integration, Transaction Processing, Performance Optimization, Job Scheduling, Financial Systems Scalability

Introduction

In today's fast-moving financial industry, clearing transactions is essential to supporting efficiency for banking and other financial services establishments. Batch control, interacting with significant quantities of financial transactions grouped into lots, is a critical component of these procedures. It allows numerous transactions to be made at pre-set times, thus accomplishing tasks ranging from credit cards to fund transfers. Batch processing is necessary because it allows the large volumes of information resulting from financial transactions to be processed fast, accurately, and in a single run. However, higher volumes or traffic pose many problems to transaction management. Among the main challenges is that much data needs to be analyzed daily. Every day, millions, if not billions, of transactions pass through financial institutions, which may occur at different times in different time zones. This makes it challenging to ensure fast and accurate transaction processing. Furthermore, preserving data accuracy and protection in such tremendous quantities is crucial since any mistake can lead to severe losses or loss of confidence in clients.

Since financial transaction systems are used in critical business processes, efficiency, scalability, and reliability are the main drivers of this kind of system. Scalability is meant to allow the system to handle the increased load with an increase in the number of transactions. Through efficiency, a firm Processes as many transactions in the shortest time possible cuts operational costs and can deliver financial services to its customers within the shortest time possible. Reliability, conversely, guarantees that the system will be available and produce accurate results during a specific period of high transaction loads or in the event of system crashes. All these factors play a vital role in strengthening the confidence of financial institutions and their customers. Though mainframes have yet to be well known, they have high-speed processing characteristics and high reliability to solve these challenges worldwide. Mainframes have been the fundamental of large and powerful transaction processing for many decades. Their capacity for data processing, their capacity for doing more than one job at a time, their security functions, and their ability to return to normal operating status after a failure make them invaluable to many firms in the financial services industry. Mainframes can handle such large transactions as needed in specific volumes, simultaneously meeting the finance subsector's security, accuracy, and velocity.

Mainframe applications in batch processing functions accrue several benefits, especially in financial systems. First, mainframes bring a reliable and scalable platform that can manage the flows of financial transactions at the necessary scale with the lowest possible risk of interruption and the highest possible speeds. Due to their affinity for parallel data processing, they can quickly help decentralize transaction processing to

Vol.10No.12(2019),92-109
DOI: https://doi.org/10.61841/turcomat.v10i12.14954

minimize bottlenecks. In addition, mainframes are safety-oriented systems that enable the storage of large amounts of data, including significant financial information, with protection from threats, including hacking, as well as from compliance with applicable regulatory standards. This article aims to identify how the scalable batch processing system, through mainframes, may enhance financial transaction systems. It seeks to explore the mainframe environment in more detail, COBOL and JCL for batch job execution, and how to approach databases such as DB2 to understand how such systems can massively improve the scalability and reliability of financial transaction processing in today's banking applications.

Understanding Batch Processing Systems

What is Batch Processing?

Batch processing is a way in which data is processed in bulk and not simultaneously as in a continuous process. In financial systems, batch processing is usually applied to solve problems such as transaction processing, account balancing, and others that can be realized simultaneously. It is a comprehensive system for handling transactions, which can be performed in bulk form at particular time intervals, generally when they are not so critical to interrupt operations. In a batch processing organization, data accumulation, processing, and storage are done in one instance and one chunk, devoid of user interaction. This is particularly evident in application areas such as bank batch processing, which would incorporate all daily transactions or account reconciliation for the month. This method is different from immediate processing, which processes transactions as they happen (Gray & Reuter, 1992). In a real-time system, when a user performs a transaction, the system also responds as soon as the transaction takes place. Real-time is essential where immediate response is needed, and output must be validated at that point of the system, like at a point of sale or ATM, to mention a few. On the other hand, batch processing is designed for operations that do not need immediate feedback but must process real-time data at a very high speed. Financial institutions usually apply batch processing tools that are not time-sensitive but demand vast amounts of effort to process.

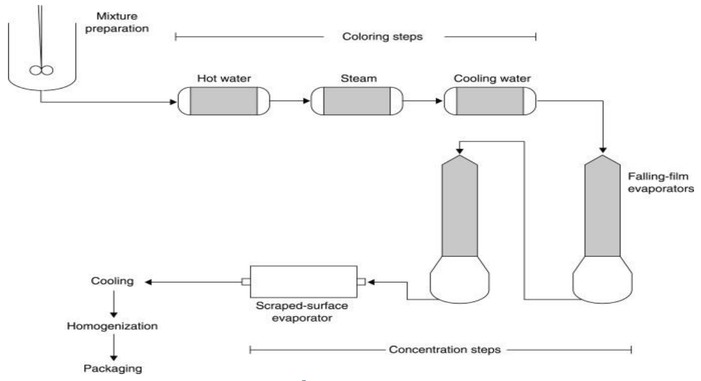

Figure 1: Batch Process - an overview

Role of Batch Processing in Financial Systems

Batch processing is essential in processing large transaction volumes in financial systems, primarily in banking and other industries. It assists in handling large streams or batches of data, which is very important in the effective functioning of the banking business. Batch processing is essential in processing repetitive tasks in a firm's operations (Cooper, 1990). especially in financial institutions where transactions are not processed individually. For instance, organizations like banks undertake large volumes of transactions, including payment, wire transfer, or computation of interest, through batch processing systems. These systems operate by accumulating transactions during a specific time and then processing them in batch form, hence minimizing frequent interaction with the systems. In contrast to online updates or payment processing on every transaction, a bank would only update all accounts or process payments or reconciliation nightly through batch processing.

Everyday batch processing use cases in finance include:

- **Account Balances:** Financial institutions employ batch processing to obtain the daily balance on accounts through all the transactions completed during the day. Different from where account balances are calculated for each transaction, which may prove very costly regarding time and other resources, the system consolidates all such data and processes it simultaneously.
- **Fund Transfers:** When money is moved between accounts or between institutions, the batch processing systems process it in batches (Barker & Rawtani, 2004). This helps ensure that all transfers are made orderly and prevents wrong figures from being recorded.

Vol.10No.12(2019),92-109
DOI: https://doi.org/10.61841/turcomat.v10i12.14954

- **Interest Calculations:** Financial institutions assess the interest rates and add them to a customer's account over a period or interval, like monthly or quarterly. In real-time, the interest rates are set to account by account. Batch systems calculate interest, and apps apply all the accounts simultaneously, generally performing the task at night.

These use cases showcase how a batch-processing scenario provides economies and guarantees that large numbers of financial transactions can be processed without overwhelming an organization's operational systems and structures or requiring sufficient user interactivity.

Challenges in Traditional Batch Processing

That said, there are several limitations associated with TBP: a number of these are particularly acute when applied to banking and large volumes of daily financial transactions. One of the major problems is how long it takes to process a large volume of information. Real-time systems take less time than batch-processing systems since data collection happens after a certain period. At some times, such as the end of the month or after financial events like the end of the fiscal year, the batch jobs can take a very long time to execute, and this makes reports and data. It also has the disadvantage of minimal processing in that it takes far longer than real-time, a significant downgrade in environments that need constant updates like finance.

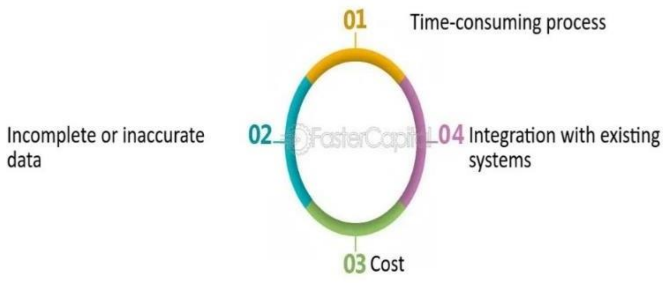
01 Time-consuming process
02 Incomplete or inaccurate data
03 Cost
04 Integration with existing systems
FasterCapital

Figure 2: Common Challenges in Batch Processing and How to Overcome Them

One of the problems relates to data accuracy. Traditional batch systems work on the principle that the data produced is only as accurate as the batch process used in their creation. Any errors that may exist cannot be detected as soon as they arise; instead, they can be discovered only after the batch job is complete. This results in incorrect values that may require tedious correction later. Data integrity in these complex applications in current legacy batch processing systems should be closely monitored, checked for errors, and further validated at every implementation stage. However, even with these preventions made, it is still possible to delay the recognition of a problem, which decreases the relevance of the system in its entirety. In addition, scalability is the primary concern of conventional batch processing systems. As financial institutions grow or the volumes of transactions per day rise, the question of scalability arises (Sompolinsky & Zohar, 2013). The prior generation of batch systems can be inflexible in accommodating the high numbers of transactions that operate in today's financial institution environments effectively, which puts pressure on scalability without necessarily increasing the operational expense or witnessing a decline in the system's performance. It is a common problem to face difficulties in scaling a traditional batch processing system since this means a lot of re-designing work that can be very costly in terms of time and money. This factor is more critical in today's fast-changing financial environment, where scale matters. The increased number of financial transactions has to be accommodated without affecting the velocity or accuracy of the financial institutions. Data Processing can handle a large amount of work that powers the batch processing system, making it necessary to address issues with scalability to meet the demands of digital banking, mobile transactions, and e-commerce. Batch processing systems that cannot scale efficiently in this sense suffer the consequences of lower and increasingly processed transaction times and increased operation costs. In financial institutions, these limitations lead to poor customer experience, enhanced error rates, and potential compliance issues.

The Mainframe Advantage

What are Mainframes?

Mainframes are large, high-capacity processing systems that have been part of the core of large business systems since the early fifties. They were first designed by the International Business Machines Corporation

Vol.10No.12(2019),92-109
DOI: https://doi.org/10.61841/turcomat.v10i12.14954

(IBM) to handle various data and supply processing power to the business, administration, and financial community. On the other hand, mainframes are designed to handle large volumes of data simultaneously. They are more reliable, scalable, and secure than personal computers or even some server systems. Traditionally, MAN has been associated with computers that, among other things, handle large amounts of data processing in organizations like banking, insurance, and telecommunication (Attewell & Rule, 1984). Mainframes have not been rendered irrelevant by distributed computing or the increasing use of cloud solutions; they retain their relevance in the specific scope in which the need for continuous computing and many transactions arises. Notably, mainframes have been extensively used in the financial service industry, given their capacity to accomplish numerous and even delicate transactions promptly and consecutively.

Figure 3: Mainframe computer

The following features characterize mainframes, unlike other computing systems. These machines are built with failovers that have multiple processors, storage systems, and paths to get data to users. They are also capable of handling transactions in parallel, which means that they can work on many works at a time without affecting the speed or accuracy of the work. Such features mark mainframes appropriate for industries that cannot afford the luxury of error or a system breakdown like monetary transactions.

Why Mainframes are Ideal for Financial Transaction Processing

Mainframes are best suited to high workloads and high-reliability networks, hence their application in financial transactions. The most significant benefit of mainframes is their computing capacity. Their execution rates are significantly high. These systems can perform millions of instructions per second, so they can be used for complex real-time calculations /processing of large data. This is more important for organizations such as banks and other financial outlets that may deal in thousands, not to mention millions, of transactions. Another critical aspect of mainframes is that they support Parallel Processing. Mainframes can help subdivide tasks to different processors and work on several transactions simultaneously. This parallel process capacity enables the system to handle many transactions at a given period of the day or month without any hitch, ensuring that as many financial transactions are processed within the shortest time possible.

Another area that also makes mainframes the preference in financial environments is the aspect of security—finance institutions deal with critical information input, which involves details of accounts and transactions. Hence, the security of information is paramount. Mainframes also have executable security features such as encrypting, controlling, and auditing an organization's confidential data to avert invasion or alteration by an unauthorized external party or internal users. Moreover, error recovery in mainframes is well done to warrant that data is well protected even when the system has failed and can be recovered quickly (Neumann, 2000). These security and reliability features ensure that people trust financial transactions and meet legal requirements. Mainframes also provide for high availability, which is critical in financial institutions that are open year-round. These systems contain redundancy and can handle faults, making them relatively low downtime. For instance, one component may be unable to operate, whereas another can come in and complete a transaction without interference. Such reliability also means it is possible to carry out transactions without interference, regardless of a problem with the hardware or software systems.

Role of COBOL and JCL in the Mainframe Batch Processing

Vol.10No.12(2019),92-109
DOI: https://doi.org/10.61841/turcomat.v10i12.14954

Due to their efficiency and reliability, even today, mainframes are coded in COBOL. This language was designed in the 50's to deal with the large-scale operations still typical of financial markets. COBOL finds wide application in performing batch processing in financial systems due to its ability to handle large data sets with great ease and speed for data transforms on those sets. For instance, COBOL is used in banking applications to perform transactions, alter account balances, and produce reports (Whitson, 2007). This is a logical language for performing business operations. Its syntax provides simple designations, reflecting its recognition of maintainable code. This is especially crucial in areas such as the financial sphere, where applications, structures, and flows must be accurate, protected, and modified to always respond to new standards. Some mainframe systems and, most importantly, DB2 can accept, process, and store transactions in COBOL programs.

Another important factor of mainframe batch processing is Wipro, Job Control Language (JCL). JCL is a type of scripting language used in designing, scheduling, and handling in-frame batch jobs. In the context of financial transaction processing, JCL has the significant function of explaining how batch jobs are executed automatically. For instance, J, L is employed to determine which COBOL programs are to execute, at what time, and in what sequence, thereby maintaining a proper order of transaction processing. JCL helps the mainframe to perform massive automated batch work without human interference, thus assuring massive consistency in its financial dealings. It also supports prioritizing the workload, hence protecting more critical jobs, such as jobs linked with real-time transaction processing. An adequate level of controlling and planning the jobs is critical concerning the actual handling of a vast number of transactions since it keeps resource consumption at an optimal level and prevents the system from overloading during the cycles of intensive workload (Datta et al, 1996).

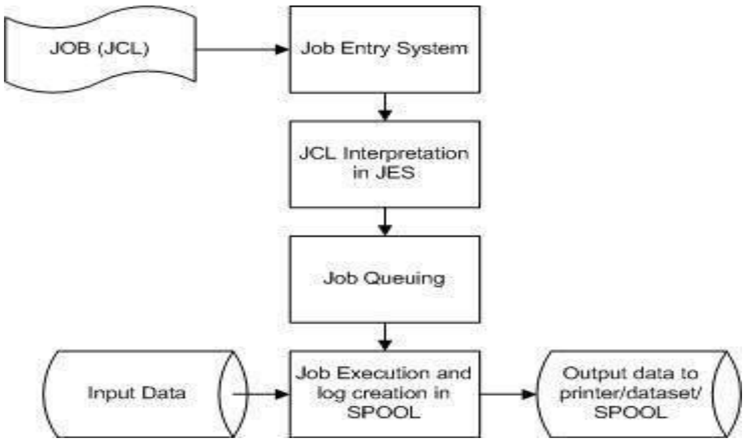
JOB (JCL)
Job Entry System
JCL Interpretation in JES
Job Queuing
Input Data
Job Execution and log creation in SPOOL
Output data to printer/dataset/ SPOOL

Figure 4: JCL - Overview

Designing Scalable Batch processing systems

Organizing efficient strategies for batch processing is essential since most applications in the present financial system have high requirements. With the rise in transaction multiples, the demand for structures capable of managing large amounts of data without sacrificing speed, dependability, or availability also rises. In this section, we will examine the concepts that should be considered when implementing an extensive bandwidth batch processing system, the methods that may be used to improve data management and retrieval procedures, and how to optimize batch planning and job running.

Key Considerations in Designing Scalable Systems

The core success factor has been the requirement of scalable batch processing systems intended to support expanding volumes of transactions. Scalability emphasizes the system's ability to either increase or decrease the capacity it uses to perform the workload optimally. For financial institutions, where the transaction numbers can vary greatly, scalability is not a nice-to-have feature but an absolute requirement.

- **Importance of scalability in handling increasing transaction volumes:** Scalability makes batch processing systems capable of accommodating an increase in both the volume of data and the level of processing of the transactions. Banking institutions may realize more demands per minute, hour, or day when they grow. A Realizable System can collect more information by having a more powerful processor, massaging more information in the same processor, or dividing the load between two or more systems or servers (Hewitt, 2010). This ability allows supporting transaction time and, at the same time, increases data volumes if necessary. One of the disadvantages is that scalability can bring processing delays, system breakdown, and the inaptitude to achieve SLAs at the time of outstanding volumetric transactions. For example, there is high traffic in some specific global markets, such as during financial market closures or promotional sales periods.

Vol.10No.12(2019),92-109
DOI: https://doi.org/10.61841/turcomat.v10i12.14954

A scalable batch processing system must also minimize the adverse impact of high load periods; the system should ideally be able to scale up its usage of resources dynamically.

- **Designing systems for load balancing and redundancy:** Load balancing and redundancy are critical tenets of any highly scalable batch-processing application. Load balancing ensures that the workload is divided among the resources in a particular system, thereby increasing the efficiency of a particular system through the servers or the processors. This helps to avoid congestion and thereby increases overall performance. For instance, if some server has low efficiency or replies to many messages, the load balancer can forward jobs to other more free servers and, in this way, no problem with a single node can slow down the whole system. Redundancy is, however, significant as far as system reliability and availability are concerned. They include mirroring essential system parts, including storage and processing units, so backups can be easily created whenever a hardware breakdown occurs. In batch processing settings, redundancy means the system can provide its services without a hitch, even when one part of the system is problematic. This decreases the chance of losing info and costly downtimes, which is critical in some financial systems where delays within perusing transactions may lead to large disasters.

Data Handling and Optimization Techniques

Data management is the basic principle of designing large-scale batch processing systems. Financial transactions involve Inputs, which must be processed, retrieved, and stored in the least amount of time using the minimum resources (DeWitt, et al, 2007).

- **Techniques for Efficient Data Storage, Retrieval and Processing:** For adequate small and large volumes of transactions, efficient data storage and retrieval are required. Indexing is helpful in databases since it increases the speed of conducting the following search on the data as it develops an index for the data. This means that, through adequately structured data, batch processes can do away with scanning a complete dataset and instead go directly to the data required for the particular batch process and significantly increase the transaction processing speed. Another method learned is data partitioning, which divides the extensive data into more traceable data sets. This makes it possible for related partitions to be worked on independently by other processors or servers and thus work concurrently. For instance, a batch of jobs in a firm that involves processing financial transactions across different regions can be partitioned so that each region's transactions file is processed in parallel and the time taken to process the whole batch is minimized.
- **Use of Indexed Data Structures, sorting and Compression Techniques:** Optimization solutions cannot be discussed separately from indexed data structures. B-tree or hash indexes, for example, can be used; these make how information is retrieved efficiently. This is obvious, especially when data is manipulated in real-time or complex queries across large databases. It is also essential for increasing batch processing performance. Data pre-processing involves arranging data before its first processing stage to fit the required company standards (Christen & Christen, 2012). For instance, suppose there are many transactions taken in a batch, and one can sort them based on the time of the transaction or the transaction size before feeding them to the system for further processing; it will be easier for the computer to handle aggregations or consolidations of the transactions to come up with summary measures.

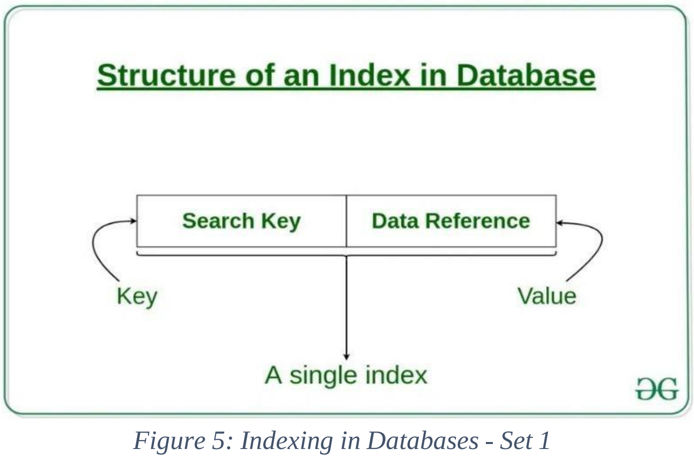

Figure 5: Indexing in Databases - Set 1

Batch Scheduling and Job Execution Strategies

Vol.10No.12(2019),92-109
DOI: https://doi.org/10.61841/turcomat.v10i12.14954

It is important to recognize that managing batch schedules and general mappings of jobs onto systems has a major impact on scalability. This paper discusses how work is prioritized, timed, and performed and indicates that the manner in which these aspects are organized influences the efficiency of a financial transaction processing system.

- **Optimizing job schedules for peak performance:** These are some of the measures most organizations apply to improve performance: s, schedules batch jobs dg the off-peak you generally, most financial systems operate with varying intensities during the day depending on time, with high intensity during the market hours or end-of-the-day settlements. Since large batch runs may cause concentration on real-time transaction processing, institutions should consider running these large batch jobs during off-hours or after business hours. Also, it is possible to regard the dependencies between jobs as direct and to provide a mechanism for scheduling job changes in response to the current load to prioritize crucial jobs with high load.
- **Job Prioritization and Resource Allocation:** Steady throughput batch jobs can be critical, while the batch-intensive type requires many resources. F redraw, the above system enables prioritization of Jobs such that those with a higher priority are run first. For instance, tasks in the finance area, some involving settlements and others related to fraud, could be prioritized over normal activities such as data backup and reporting. Resource management provisions guarantee that work is offered fundamental physical resources, including a fair share of CPU time, memory, and disk space as required (Gulati, et al, 2012). For example, during burst working periods, different tasks that need more CPU resources can be granted more CPU cores or RAM for their fast execution. In an environment that uses multiple processors or cloud computing, resources can be intermittently assigned for every job to get the needed resources without burdening the system.

Implementing Batch Processing with COBOL and JCL

COBOL for Batch Job Development

These include COBOL, which has been the basis of numerous financial systems for years. COBOL today is still the language used in many industries for their batch job needs, especially the financial sectors because it performs well when used to handle large volumes of data, it is stable, and its relatively simple coding language is ideal for use when dealing with complex and detail-oriented projects, such as in the financial business. Furthermore, since COBOL best supports legacy systems and databases, such as DB2, it remains relevant in today's multinational companies' huge-scale financial transaction processing workflow. COBOL is still widely used in financial systems, among other reasons, because it performs well in batch jobs that involve large amounts of data. Under Batch Processing, COBOL codes programs that perform a sequence of operations, such as transforms on data, transactions, or computations on many records that may not be processed in direct contact by users or clients. COBOL is designed to optimize for such uses. Thus, there is fast batch job computation to improve execution time (Hummel et al, 2010). Regarding code structure, COBOL programs for batch processing typically consist of several key sections: program Identification, Program Metadata, System Environment Definition identified by the Environment Division, variables and files declared in the Data Division, and the Program Logic defined in the Procedure Division.

A simple example of a COBOL program for processing financial transactions might look as follows:

Table 1: A simple example of a COBOL program for processing financial transactions

```
            cobol
            IDENTIFICATION DIVISION.
            PROGRAM-ID. TransactionProcessing.

            ENVIRONMENT DIVISION.
            INPUT-OUTPUT SECTION.
            FILE-CONTROL.
               SELECT TRANSACTION –FILE ASSIGN TO 'TRANSACTIONS.DAT'
                  LINE SEQUENTIALITY APPLICABLE TO ORGANIZATION OF IT IS FOLLOWING:01
   TOTAL-AMOUNT PIC 9(9)V99 VALUE ZERO.ng with COBOL and JCL
         COBOL for Batch Job Development
```

COBOL has been a cornerstone of financial systems for decades. Despite the rise of newer programming languages, COBOL remains the language of choice for batch job development in many industries, particularly

Vol.10No.12(2019),92-109
DOI: https://doi.org/10.61841/turcomat.v10i12.14954

finance. Its longevity in the financial sector is due to its robust handling of large data volumes, stability, and straightforward syntax, which is well-suited for financial applications' complex and precise nature. Moreover, COBOL's ability to interact with legacy systems and databases, like DB2, makes it an integral part of enterprise-level financial transaction processing. One of the main reasons COBOL continues to dominate financial systems is its efficiency in handling batch jobs that process large volumes of data. In batch processing, COBOL writes programs that execute operations—such as data transformations, transactions, or calculations—on extensive records, often without real-time user interaction. COBOL is optimized for such tasks, enabling high-performance processing that ensures quick execution of batch jobs.

In terms of code structure, COBOL programs for batch processing typically consist of several key sections: the Identification Division, which includes program metadata; the Environment Division, which defines system resources (e.g., files); the Data Division, where variables and file structures are declared; and the Procedure Division, where the actual logic of the program is written.

A simple example of a COBOL program for processing financial transactions might look as follows:

Table 2: A simple example of a COBOL program for processing financial transactions

```
cobol
IDENTIFICATION DIVISION.
PROGRAM-ID. TransactionProcessing.

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
   SELECT TRANSACTION-FILE ASSIGN TO 'TRANSACTIONS.DAT'
      ORGANIZATION IS LINE SEQUENTIAL.
DATA DIVISION.
FILE SECTION.
FD  TRANSACTION-FILE.
01  TRANSACTION-RECORD.
   05  TRANSACTION-ID       PIC 9(5).
   05  AMOUNT               PIC 9(9)V99.
   05  DATE                 PIC X(10).

WORKING-STORAGE SECTION.
01 TOTAL-AMOUNT           PIC 9(9)V99 VALUE ZERO.

PROCEDURE DIVISION.
OPEN INPUT TRANSACTION-FILE.
beginTransaction(Read Transaction-File INTO Transaction-Record)
   AT END
      DISPLAY 'End of File Reached'
   NOT AT END
      ADD AMOUNT TO TOTAL-AMOUNT
      DISPLAY 'Transaction ID: ' TRANSACTION-ID
      DISPLAY 'Amount: ' AMOUNT
      DISPLAY 'Total Amount Processed: ' TOTAL-AMOUNT
END-READ.
CLOSE TRANSACTION-FILE.
STOP RUN.
```

The following is a simple COBOL program that reads a file full of financial transaction data and adds the value of every transaction to the sum total. The program illustrates how easy COBOL is for data handling and how it applies to external file resources, which is a major feature in batch-type operations in financial areas.

JCL for Job Scheduling is a technique used to schedule jobs.

JCL is necessary for preparing, submitting, and tracking batch processes in mainframe computing systems (Collins & Hester, 2011). Batch file scripting language is another language used for defining jobs, the resources necessary for the job, the input and output files, and the steps to be followed. As in most traditional

Vol.10No.12(2019),92-109
DOI: https://doi.org/10.61841/turcomat.v10i12.14954

financial transaction processing systems, JCL scripts are used to time and manage COBOL batch programs running at certain times and with specific system resources. In other words, the JCL script is the prescription of the mainframe's working for a particular job to run, say a COBOL program, and outlines some datasets to be used, some job parameters to be set, and the working environment for the particular job. For instance, the JCL script defines the input transaction and output files and the COBOL program that should be run. They also include the management of resources such as memory allocation, dataset formation, and files during batch jobs.

Here is a basic example of a JCL script that runs a COBOL batch job:

Table 3: A basic example of a JCL script that runs a COBOL batch job

```
jcl
>//TRANSACTION          JOB          (ACCTNUM),Batch          Processing          [-]>//
CLASS=A;MSGCLASS=X;NOTIFY=&SYSUID//COPY               FROM               -               DD
DSN=TRANSACTIONS.DAT,DISP=SHR>OUTFILE               DD               DSN=OUTPUT.TXT,
DISP=(NEW,CATLG,DELETE),>// UNIT=SYSDA, Space : ( CYL ,( 1, 1 ).Processing with COBOL and JCL
    COBOL for Batch Job Development
```

COBOL has been a cornerstone of financial systems for decades. Despite the rise of newer programming languages (Lovrenčić et al, 2009). COBOL remains the language of choice for batch job development in many industries, particularly finance. Its longevity in the financial sector is due to its robust handling of large data volumes, stability, and straightforward syntax, which is well-suited for financial applications' complex and precise nature. Moreover, COBOL's ability to interact with legacy systems and databases, like DB2, makes it an integral part of enterprise-level financial transaction processing. One of the main reasons COBOL continues to dominate financial systems is its efficiency in handling batch jobs that process large volumes of data. In batch processing, COBOL writes programs that execute operations—such as data transformations, transactions, or calculations—on extensive records, often without real-time user interaction. COBOL is optimized for such tasks, enabling high-performance processing that ensures quick execution of batch jobs.

In terms of code structure, COBOL programs for batch processing typically consist of several key sections: the Identification Division, which includes program metadata; the Environment Division, which defines system resources (e.g., files); the Data Division, where variables and file structures are declared; and the Procedure Division, where the actual logic of the program is written.

A simple example of a COBOL program for processing financial transactions might look as follows:

Vol.10No.12(2019),92-109
DOI: https://doi.org/10.61841/turcomat.v10i12.14954

Table 4: example of a COBOL program for processing financial transactions

```
cobol
IDENTIFICATION DIVISION.
PROGRAM-ID. TransactionProcessing.

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT TRANSACTION-FILE ASSIGN TO 'TRANSACTIONS.DAT'
        ORGANIZATION IS LINE SEQUENTIAL.

DATA DIVISION.
FILE SECTION.
FD  TRANSACTION-FILE.
01  TRANSACTION-RECORD.
    05  TRANSACTION-ID       PIC 9(5).
    05  AMOUNT               PIC 9(9)V99.
    05  DATE                 PIC X(10).

WORKING-STORAGE SECTION.
01  TOTAL-AMOUNT             PIC 9(9)V99 VALUE ZERO.

PROCEDURE DIVISION.
OPEN INPUT TRANSACTION-FILE.
READ TRANSACTION-FILE INTO TRANSACTION RECORD
    AT END
        DISPLAY 'End of File Reached'
    NOT AT END
        ADD AMOUNT TO TOTAL-AMOUNT
        DISPLAY 'Transaction ID: ' TRANSACTION-ID
        DISPLAY 'Amount: ' AMOUNT
        DISPLAY 'Total Amount Processed: ' TOTAL-AMOUNT
END-READ.
CLOSE TRANSACTION-FILE.
STOP RUN.
```

This basic COBOL program processes a file containing financial transaction data, adding the amount of each transaction to a running total. The program demonstrates the simplicity of COBOL for data handling and its ability to work with external file resources, a critical aspect of batch processing in financial systems.

Job Control Language (JCL) for Job Scheduling

Job Control Language (JCL) is essential for managing batch jobs on mainframe systems. It is a scripting language used to define and manage job execution, specifying the required resources, input and output files, and the sequence of operations. In a typical financial transaction processing system, JCL scripts are used to schedule and control the execution of COBOL batch programs, ensuring they run at the appropriate times and with the right system resources. A JCL script instructs the mainframe on executing a specific job, such as a COBOL program, and defines the required datasets, job parameters, and execution environments (Richardson, 2008). For example, a JCL script might specify the input transaction file, output file, and the COBOL program to execute. It also handles resources like memory allocation, dataset creation, and file management during the batch job.

Here is a basic example of a JCL script that runs a COBOL batch job:

Vol.10No.12(2019),92-109
DOI: https://doi.org/10.61841/turcomat.v10i12.14954

Table 5: basic example of a JCL script that runs a COBOL batch job

```
Jcl
//TRANSACTION JOB (ACCTNUM),' Batch Processing,'
//         CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//STEP1     EXEC PGM=COBOLPROGRAM
//TRANFILE  DD  DSN=TRANSACTIONS.DAT,DISP=SHR
//OUTFILE   DD  DSN=OUTPUT.TXT,DISP=(NEW,CATLG,DELETE),
//            UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSOUT    DD  SYSOUT=*
//SYSIN     DD  *
  COBOL PROGRAM PARAMETERS
/*
```

With the help of this JCL script, a job called TRANSACTION is created, in which the COBOL program COBOLPROGRAM is executed at the first step (STEP1). It takes input and output file datasets with the TRANFILE and OUTFILE, defines the system requirements, and controls the output. By setting these parameters, JCL controls and smoothes the running of COBOL programs as batch jobs through the automation that has been provided for them. Here is more information about specific ways of approaching the acceleration of JCL for quicker capability to transact in a business operating environment: For example, giving the appropriate priority to the distribution of job resources in order to avoid bottlenecks, controlling the use of reasonable memories and CPU and accurately determining the start-stop sequence of multiple batch jobs have a significant impact on improving the comprehensive performance of large systems. Also, performing file management efficiently in JCL, for example, reducing I/O operations and maximizing parallel processing, will lead to faster transaction processing time.

Integration with Mainframe Databases (e.g., DB2)

In financial systems, it is crucial how COBOL batch programs can be interfaced with mainframe databases, including DB2. DB2 is a relational database management system typically installed on mainframes to store financial transactions (Kooijmans et al, 2012). It is scalable, reliable, and cost-effectively processes complex queries. SAP can be implemented in COBOL programs through SQL, which is integrated into the maintenance of DB2. This enables COBOL programs to use SQL statements to retrieve, amend, and even erase records in a DB2 database. This integration is critical in a financial transaction system because it can handle large amounts of data while simultaneously providing checks and balances in the entire queue.

Here is an example of how COBOL can be used to retrieve data from a DB2 database:

Vol.10No.12(2019),92-109
DOI: https://doi.org/10.61841/turcomat.v10i12.14954

Table 6: an example of how COBOL can be used to retrieve data from a DB2 database

```
COBOL
EXEC SQL
   DECLARE C1 CURSOR FOR CURSOR1
   WHERE SELECT TRANSACTION_ID, AMOUNT FROM TRANSACTIONS
   WHERE DATE = :CURRENT-DATE
END-EXEC.

EXEC SQL
OPEN C1
END-EXEC.

PERFORM UNTIL SQLCODE NOT = 0
   EXEC SQL
FETCH C1 INTO: TRANSACTION-ID, AMOUNT
   END-EXEC.
   ADD AMOUNT TO TOTAL-AMOUNT
END-PERFORM.

EXEC SQL
   CLOSE C1
END-EXEC.
```

In this example, COBOL uses SQL within that program to talk to DB2 engines. It uses a TRANSACTIONS table to retrieve the transaction IDs and the amount of these transactions for a particular date and sum them. This integration enables organizations to perform data usage features on COBOL programs, from archives to altered financial transactions. To enhance this integration, developers, for instance, index the DB2 tables to enable faster data retrieval and ensure that the SQL queries used consume as little resource as possible. Also, controlling connections with databases and avoiding as many database calls as possible will ease the transaction latency problem.

Optimizing Batch Processing Systems

Such systems are used mainly for processing a high volume of financial transactions in a poorly designed medium. However, as the number of transactions rises, the efficiency of the systems becomes paramount, facilitating high traffic without affecting the speed and effectiveness of the particular system. This part of the dissertation outlines optimization techniques for system performance, load distribution, resource utilization, and fault tolerance measures for scalable batch processing systems.

The Performance Optimization techniques

The first step is determining and removing the factors that negatively affect system performance. The constraints may appear during data collection, computation, or job processing, depending on the system's design. To determine these bottlenecks, the total system utilities can be utilized to check the consumption of resources for CPU, memory, and disk, then be able to notice where there are delays. For instance, when impounded, the repetitive job may take a long time to process; reviewing the code, logic, or even the database job used in the process is essential. When bottlenecks are defined, many methods can be employed to enhance efficiency. One method discussed is data indexing. When data processing is associated with vast amounts of data, indexing helps to find necessary data quickly instead of scrolling through the records (Umbrich et al, 2011). Moreover, other tactics can be applied to increase the performance of queries when interacting with the DB2 DBMS, such as rewriting SQL queries so that unwanted JOINs or subqueries are removed.

Another optimization technique the scheduler must use is reducing I/O wait time. Input-output operations to the disk involving read or write operations are usually time-consuming within batch jobs. Decreasing the number of I/O operations or using faster media (e.g., SSDs) can significantly improve their rate. Further, there is parallel processing, where the work is divided into smaller manageable parts and processed on different CPUs simultaneously. This makes workload pro-options on I/O. Another indispensable goal is optimizing the CPU consumption rate within a system. This can be followed up by making code and logic faster to perform operations. Packing more data in a single signal can, for instance, reduce the number of iterations in loops or optimize the

Vol.10No.12(2019),92-109
DOI: https://doi.org/10.61841/turcomat.v10i12.14954

work of recursive functions, thus reducing CPU usage (Panda et al, 2001). Also, loading the necessary data rather than the datasets into memory helps cut down the CPU loading and resource distribution.

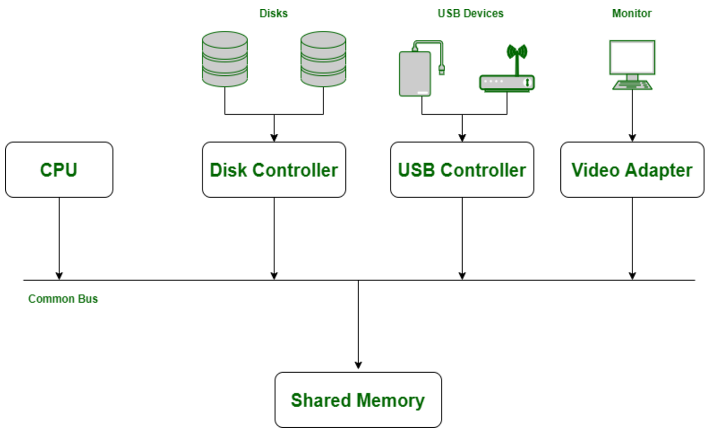

Figure 6: I/O scheduling in Operating Systems

Load Balancing and Resource Allocation

As the demand for batch processing increases, load balancing naturally becomes critical. Load balancing makes it possible to distribute workloads optimally across several processors or servers, so no single resource will always be busy while others have little to do. This is particularly the case in financial transactions, where high transfer volumes have to be facilitated during particular core business surges. Dynamic load balancing acts in real-time, moving jobs around based on how well the system runs and its workload. Algorithms track the system's use of resources and redirect to servers with higher capacity. This can also be practiced to help in distributive processing to ensure that all the central processing units process less workload than other systems, thus reducing delays. The appropriate distribution of resources is central to system efficacy. Through the flexibility of assigning resources, fancily CPU time, memory, and storage, for the elapsed workload, systems can provide or sustain optimal expectations during times of paralysis. For instance, additional resources can be afforded to handle batch jobs during busy periods of many customer transactions (Buczynski, 2013). During periods of low customer traffic, such resources can be released for other activities or maintenance exercises. Resource sharing may be used for virtualization purposes for systems located on mainframes. With VMs, users can be provisioned and de-provisioned depending on real-time workloads so that more resources are allocated whenever there is high demand and less when low demand is needed.

Error Handling and Recovery Mechanisms

It is essential to incorporate error-trapping and correction provisions in any batch processing system. Batch functions require processing sizable data, and aberrations like data incoherence, transmission problems, or equipment breakdowns are possible at any time. If error resolution is weak, these cases can result in wrong outputs, slow execution, or system failure. Validation is the first step in detecting errors appropriately. Various kinds of data validation checks can be implemented into batch jobs to ensure inputs into the batch processes meet the required format, range, or other constraints before processing. For instance, in the financial context, completed transaction details can be checked and compared with the reference data to include a complete and accurate account number, the amount value, and the date before the transaction.

After an error has been identified, error control measures are initiated. For example, in this COBOL system, conditional statements can guide the system to write errors, terminate, or try for a prescribed recovery. Batch jobs should have provisions for their failure by enabling the system to handle other jobs concurrently in case one fails. Furthermore, the recovery mechanisms are critical to enhance the system's ability to come to a normalized state once it has failed. This gets done in batch systems using what is referred to as checkpoints; checkpointing means copying in some form of way or another the status of a batch job at periods that can be a few minutes apart so that if for any reason a failure occurs, then the job is not obliged to start afresh from scratch but rather resumes from the last point of the checkpoint. A second significant recovery technique is the transaction log, in which every transaction process is documented. This facilitates the reconstruction of the transactions when the system has crashed. It is, therefore, essential to consider data integrity issues when implementing error recovery. For instance, financial operations should be performed so that any modifications made to the database are performed as atomic – in other words, all of it or none (Cooper & James, 2009). This can be conducted through the application transaction controls in databases like DB2 to ensure the integrity of the financial data regarding the ACID properties – Atomicity, Consistency, Isolation, and Durability – are engaged.

Vol.10No.12(2019),92-109
DOI: https://doi.org/10.61841/turcomat.v10i12.14954

Case Study: Successful Implementation in Financial Institutions

Overview of the Case Study

In the context of this case study, the topic focuses on designing scalable batch-processing systems for significant financial organizations that process a vast number of daily transactions. This way, this institution, a leading bank in the region (which will hereafter be called "FinancialCo" for identification purposes), was overwhelmed in its ability to handle the increasing number of financial transactions. Since the number of transactions was growing, the load and requirements on the inherited systems that had been introduced significantly enhanced the speed and capacity for real-time. Based on the initial case, since the volume of transactions that occurred, the load that has arisen requires several additional comments. To overcome these difficulties, FinancialCo decided to transform its transaction processing system using a package based on the mainframe embodying COBOL and JCL to improve the critical parameter of batch processing—scalability. The purpose was to construct a scalable system capable of processing millions of transactions per day, the stability and security of which are crucial for banking activities.

Challenges Faced and Solutions Implemented

While undertaking the initial implementation stage, FinancialCo faced some problems typical of most huge batch-processing environments (Sriram, 2000). One of the most significant concerns was the need for more links to the client's extensive existing systems. The legacy architecture that needed to be built to accommodate present transaction traffic frequently encountered throughput issues, including during heavy transaction times. This led to poor processing times, increased system downtime, and even slowed down transaction updates, which affected the bank and the customers. Another problem to identify was keeping financial transactions safe and secure, which is critical in any banking organization. The old system needed to be optimized to efficiently process demanding, secure, large volumes of data. Also, the bank needed a tool that could work with the DB2 databases used by the organization for better data handling and easy data retrieval.

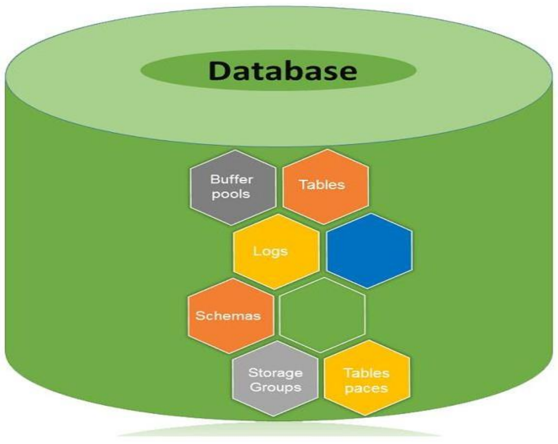

Figure 7: DB2 databases

To combat these problems, FinancialCo started transferring their batch processing workloads to a mainframe platform by implementing the COBOL language for transaction processing and job control language for compiling job flows. The choice of COBOL and JCL as key technologies was deliberate: these… technologies are relatively conservative and optimized for transaction-intensive, high-reliability financial settings. The bank also used many intelligent methods of data management and indexing to improve the enhanced system's performance. Other mainframe-based solutions include resource utilization, which could be enhanced by improved load scheduling to perform the jobs using efficient work schedules with little or no delay and less strain on the mainframe. In particular, the integration with DB2 has enabled the batch system to read data fast and process it in a robust and error-free method on a sizeable periodical basis (Gruhl, et al, 2004).

Results and Outcomes

Adopting the MBS to perform batch processing has enhanced several aspects of performance (Hu, et al, 2013). The most significant result was the improvement in the time taken to process transactions by a large margin. Before implementing the new financial system, the company's batch processing system needed to be updated and faster to manage large numbers of transactions, meaning that the transactions would sometimes slow down. It was established that since the installation of the mainframe system, the comprehensiveness of transaction processing time was cut by 40% despite the peak traffic. This enhancement increased the speed of handling transactions, not

Vol.10No.12(2019),92-109
DOI: https://doi.org/10.61841/turcomat.v10i12.14954

to mention the overall satisfaction of the customers they dealt with. By extending system capacities, FinancialCo generated more transactions without worrying too much about the system's efficiency. The identified IT benefits were Increasing the customer base, dealing with higher volumes of work without issues, and ensuring the bank's growth. Indeed, the new system accommodates up to 10 million transactions per day, something the previous system could hardly do.

The other hoped-for benefit attained soon after the system's implementation was reliability. Some of these attributes included Redundancy and fault tolerance that the mainframe system provided, which made the transaction processing operation of the bank remain highly available during the operational period, even in case of system upgrades or maintenance windows. This helped to cut down system downtime and make it possible for critical banking operations to run as and when they are supposed to. Therefore, the bank managed to announce a system uptime of 99.99%, which in turn increased customer confidence and satisfaction. However, apart from these technical enhancements, FinancialCo, unlike other companies, is defined by its performance by the EPS lift. FinancialCo determined numerous long-term cost savings. This kind of batch processing lowered the operation cost implication by the amount of resources used, such as CPU time and memory needed. Also, since the mainframe system is reliable and has low maintenance, the overall maintenance cost for the bank was reduced so that IT could work on other vital projects (Birman et al, 2011).

Future Trends and Innovations in Batch Processing

Emerging Technologies in Batch Processing

The traditional system of batch processing systems is changing significantly, mainly due to cloud computing practices and hybrid cloud platforms. Classically, mainframe environments have only been restricted to financial transaction processing requirements because of their scalability and dependability. Nonetheless, cloud technologies open new opportunities for optimizing batch processing. Cloud computing allows batch jobs to occur on the cloud or a company's cloud-based hybrid system, meaning that organizations do not need to purchase on-premise equipment to accommodate their peaks in resource demand. This flexibility makes it easy for businesses to manage peak loads without compromising performance or investing much money in infrastructure. The hybrid cloud deployment strategy especially allows the keeping of legacy mainframe systems for the most critical and time-sensitive tasks while using secondary and less critical ones in the cloud (Schulz, 2009). For example, a bank can transfer or migrate those jobs to a cloud environment while working on the batch jobs that can be processed later, say they are reconciliations at the end of the day or month. This approach helps to achieve continuity between the need to protect and preserve the traditional mainframes and simultaneously leverage the promises of scale, cost, and flexibility of the cloud.

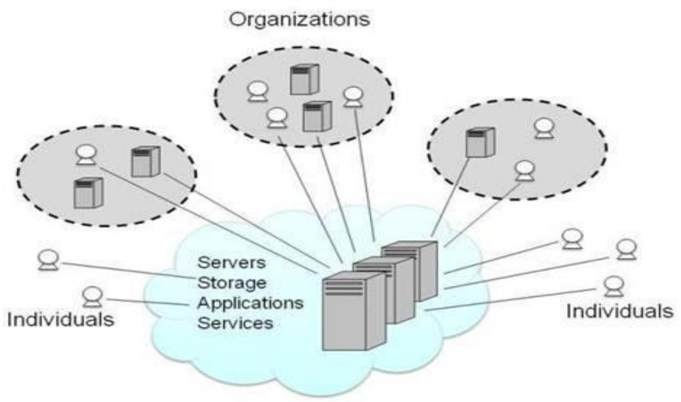

Figure 8: Cloud Computing for Business – What is Cloud?

Apart from cloud technologies, AI and ML are assuming a more substantial and essential role in the future of batch processing. The use of AI and ML shows that there is an opportunity to increase the efficiency of scheduling batch jobs as part of workload control and estimate potential periods for job processing (Bitirgen et al, 2008). For instance, patterns of previous transaction activities can be used in developing machine learning algorithms that predict the right time to process a large volume of tasks so that job scheduling is not compromised for longer durations. Therefore, AI can also help estimate errors and faults in systems that engage in batch processing. The best part of proactive measurement through machine intelligence is observing job execution and system activities, which will identify such failures or susceptibilities before they occur. This predictive capability prevents failures in an organization's environment and improves reliable transaction processing research. These

Vol.10No.12(2019),92-109
DOI: https://doi.org/10.61841/turcomat.v10i12.14954

technologies are expected to grow more integrated into the batch processing systems, increasing the efficiency, scalability, and reliability of the pace of financial transaction processing.

The Future of Mainframes in Financial Transaction Systems

Mainframes are likely to remain around for quite some time despite the emergence of cloud and advanced intelligence technologies. This is true since mainframes offer unparalleled reliability, security, and processing power and are suitable for dealing with large volumes of transactions processed in financial institutions. Nevertheless, the scale of interest in mainframes is going to change in the near future. Rather than continuing as utterly disconnected islands of computing, mainframes will be tied into cloud-based structures, making up a dual-model entity. Many future trends are being forecasted for mainframes in financial systems, indicating that these systems will be adapting to future improvements in the speed and functionality of the mainframes. Mainframes are being enhanced in the current generation to accommodate other technologies like containerization and microservices, enabling them to interface with cloud solutions openly. Thus, mainframes will become the cornerstone of the growth of financial transaction systems, providing scalability and security as the finance industry goes through digitalization (Picot et al, 2008). In contrast, cloud solutions enable financial businesses to adapt quickly and spend less on IT development. A potential improvement in the mainframe is improved processor speed and memory organization I/O capacity. This will also help mainframes deal with rising transactional traffic and more complex work, such as financial processing. Moreover, performance will be improved by integrating AI optimization techniques to optimize processing approaches in real-time to respond to the existing system state and workload.

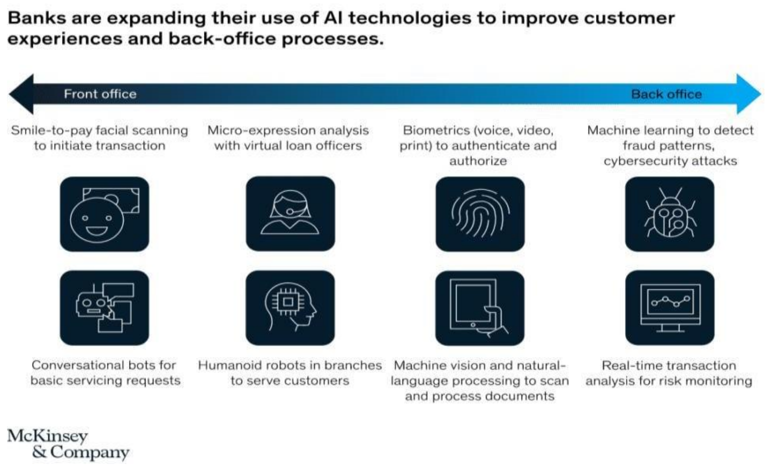
Banks are expanding their use of AI technologies to improve customer experiences and back-office processes.
Front office
Back office
Smile-to-pay facial scanning to initiate transaction
Micro-expression analysis with virtual loan officers
Biometrics (voice, video, print) to authenticate and authorize
Machine learning to detect fraud patterns, cybersecurity attacks
Conversational bots for basic servicing requests
Humanoid robots in branches to serve customers
Machine vision and natural-language processing to scan and process documents
Real-time transaction analysis for risk monitoring
McKinsey & Company

Figure 9: How Cloud Computing and AI are transforming the Financial Industry

Conclusion

To realize modern financial institutions' scalability in transaction processing with precise performance, Mainframes are vital for developing tackle able batch processing systems. This article stresses several key points: scalability, performance level, and strong compatibility with COBOL, JCL, DB2, and other technologies. Scalability is the most fundamental of all the principles of designing for batch processing. As companies grow, financial institutions must handle more volumes of transactions. Mainframes, due to their inherent high processing capability and scalability, parallel computing, and resource management capability, are well suited for these requirements (Yeo et al, 2006). Best practices like data management and job scheduling ensure that applications can be handled appropriately during heavy traffic, such as other conjugate solutions. Both JCL for job control and DB2 databases or other commercial databases provide an opportunity to increase the running speed of COBOL programs and thus improve system performance. Hence, despite its antiquity, the language is still widely used in batch COBOLs because of its integrity and effectiveness in conducting complicated financial operations. Together with JCL, which provides the framework for job execution and scheduling, these technologies facilitate the smooth running of transactions on the Mainframes. Also, DB2 helps integrate Mainframe Systems to facilitate efficient data processing and storage of transaction data. When these technologies are used together with efficient job execution strategies, an effective and high-performing financial system with growth capability is achieved.

The most inevitable is that mainframes will remain indispensable in worldwide financial transaction processing. Nevertheless, with cloud computing and many other advanced technologies still available, a Mainframe remains a great option if you need reliability, security, and sheer processing power. Financial organizations' high traffic and high-velocity requirements keep performance a top priority without risking security

Vol.10No.12(2019),92-109
DOI: https://doi.org/10.61841/turcomat.v10i12.14954

or customer data (Salsburg et al, 2003). Mainframe systems offer the reliability required in these situations. Specifically, this is why scalable batch processing systems must remain on the agenda of financial institutions as they develop and evolve in the context of rapidly provisioning more digital services. Introducing proactive Mainframe building blocks advances institutions from pressing challenges into positive outcomes, making them operationally sound and prepared for exponential transactionality and new technological systems, with confidence gained in their competitiveness and efficiency. In this way, financial institutions will be more conducive to serving future growth needs and sustaining business efficiency.

References;

1. Attewell, P., & Rule, J. (1984). Computing and organizations: What we know and what we don't know. *Communications of the ACM*, *27*(12), 1184-1192.
2. Barker, M., & Rawtani, J. (2004). *Practical batch process management*. Elsevier.
3. Birman, K. P., Ganesh, L., & Van Renesse, R. (2011, April). Running smart grid control software on cloud computing architectures. In *Workshop on Computational Needs for the Next Generation Electric Grid, Cornell University*.
4. Bitirgen, R., Ipek, E., & Martinez, J. F. (2008, November). Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach. In *2008 41st IEEE/ACM International Symposium on Microarchitecture* (pp. 318-329). IEEE.
5. Buczynski, B. (2013). *Sharing is good: How to save money, time and resources through collaborative consumption*. New Society Publishers.
6. Christen, P., & Christen, P. (2012). Data pre-processing. *Data matching: Concepts and techniques for record linkage, entity resolution, and duplicate detection*, 39-67.
7. Collins, R., & Hester, D. (2011). Mainframe Alternative Reference Implementation Redmond MTC.
8. Cooper, J., & James, A. (2009). Challenges for database management in the internet of things. *IETE Technical Review*, *26*(5), 320-329.
9. Cooper, R. G. (1990). Stage-gate systems: a new tool for managing new products. *Business horizons*, *33*(3), 44-54.
10. Datta, A., Mukherjee, S., Konana, P., Viguier, I. R., & Bajaj, A. (1996). Multiclass transaction scheduling and overload management in firm real-time database systems. *Information Systems*, *21*(1), 29-54.
11. DeWitt, D. J., Robinson, E., Shankar, S., Paulson, E., Naughton, J., Royalty, J., & Krioukov, A. (2007). *Clustera: an integrated computation and data management system*. University of Wisconsin-Madison Department of Computer Sciences.
12. Gray, J., & Reuter, A. (1992). *Transaction processing: concepts and techniques*. Elsevier.
13. Gruhl, D., Chavet, L., Gibson, D., Meyer, J., Pattanayak, P., Tomkins, A., & Zien, J. (2004). How to build a WebFountain: An architecture for very large-scale text analytics. *IBM Systems Journal*, *43*(1), 64-77.
14. Gulati, A., Holler, A., Ji, M., Shanmuganathan, G., Waldspurger, C., & Zhu, X. (2012). Vmware distributed resource management: Design, implementation, and lessons learned. *VMware Technical Journal*, *1*(1), 45-64.
15. Hewitt, C. (2010). Actor model of computation: scalable robust information systems. *arXiv preprint arXiv:1008.1459*.
16. Hu, Y., Ma, H., & Shi, H. (2013). Enhanced batch process monitoring using just-in-time-learning based kernel partial least squares. *Chemometrics and Intelligent Laboratory Systems*, *123*, 15-27.

Vol.10No.12(2019),92-109
DOI: https://doi.org/10.61841/turcomat.v10i12.14954

17. Hummel, B., Juergens, E., Heinemann, L., & Conradt, M. (2010, September). Index-based code clone detection: incremental, distributed, scalable. In *2010 IEEE International Conference on Software Maintenance* (pp. 1-9). IEEE.
18. Kooijmans, A. L., Ramos, E., De Greef, N., Delhumeau, D., Dillenberger, D. E., Potter, H., & Williams, N. (2012). *Transaction Processing: Past, Present, and Future*. IBM Redbooks.
19. Lovrenčić, A., Konecki, M., & Orehovački, T. (2009). 1957-2007: 50 years of higher order programming languages. *Journal of Information and Organizational Sciences*, *33*(1), 79-150.
20. Neumann, P. G. (2000). Practical architectures for survivable systems and networks. *Prepared by SRI International for the US Army Research Laboratory*.
21. Panda, P. R., Catthoor, F., Dutt, N. D., Danckaert, K., Brockmeyer, E., Kulkarni, C., ... & Kjeldsberg, P. G. (2001). Data and memory optimization techniques for embedded systems. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, *6*(2), 149-206.
22. Picot, A., Reichwald, R., & Wigand, R. (2008). The Potential of Information and Communication Technology for Corporate Development. *Information, Organization and Management*, 115-182.
23. Richardson, C. L. (2008). *COBOL and Visual Basic on. NET: A Guide for the Reformed Mainframe Programmer*. Apress.
24. Salsburg, M. A., Cotugno, L., Barrish, S., Leuthen, C., & Freeman, R. (2003). Mainframe Scalability in the Windows Environment. In *Int. CMG Conference* (pp. 209-215).
25. Schulz, G. (2009). StorageIO Comments and Feedback for EPA Energy Star® for Enterprise Storage Specification.
26. Sompolinsky, Y., & Zohar, A. (2013). Accelerating bitcoin's transaction processing. fast money grows on trees, not chains. *Cryptology ePrint Archive*.
27. Sriram, M. S. (2000). 1. Financial Cooperatives in Quebec, Canada: A Study of the Desjardins Movement. *JOURNAL OF RURAL DEVELOPMENT-HYDERABAD-*, *19*(2), 161-184.
28. Umbrich, J., Hose, K., Karnstedt, M., Harth, A., & Polleres, A. (2011). Comparing data summaries for processing live queries over linked data. *World Wide Web*, *14*, 495-544.
29. Whitson, G. (2007). From advanced COBOL to data, file and object structures. *Journal of Computing Sciences in Colleges*, *22*(5), 39-45.
30. Yeo, C. S., Buyya, R., Pourreza, H., Eskicioglu, R., Graham, P., & Sommers, F. (2006). Cluster computing: High-performance, high-availability, and high-throughput processing on a network of computers. *Handbook of Nature-Inspired and Innovative Computing: Integrating Classical Models with Emerging Technologies*, 521-551.