Received 22 March 2024, accepted 4 May 2024, date of publication 13 May 2024, date of current version 20 May 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3399812

RESEARCH ARTICLE

RMPF: Real-Time Message Processing Framework on Multi-Service Domains

KYO-EUN KIM[1], SEONG-SOO HAN[2], AND CHANG-SUNG JEONG[3], (Member, IEEE)
[1]Visual Information Processing, Korea University, Seoul 02841, Republic of Korea
[2]Division of Liberal Studies, Kangwon National University, Samcheok-si 25947, Republic of Korea
[3]Department of Electrical Engineering, Korea University, Seoul 02841, Republic of Korea

Corresponding authors: Seong-Soo Han (sshan1@kangwon.ac.kr) and Chang-Sung Jeong (csjeong@korea.ac.kr)

This work was supported in part by the 2023 Research Grant from Kangwon National University; in part by the Artificial Intelligence Industrial Convergence Cluster Development Project funded by the Ministry of Science and ICT (MSIT), South Korea; and in part by Gwangju Metropolitan City.

ABSTRACT In the world of real-time computing, there is a demand for platforms that deliver both speed and efficiency while conserving hardware resources. High reliability applications typically use dedicated platforms for their inherent reliability, but they are more expensive. On the other hand, traditional general-purpose platforms are cheaper but suffer from wasted hardware resources when processing messages, hindering distributed processing and process monitoring. To address these limitations, this paper presents the Real-time Message Processing Framework on Multi-service Domains (RMPF), an innovative integrated platform for real-time distributed message processing on multi-service domains. The platform supports a scalable and flexible multi-layer architecture by providing modularity that allows each layer to be designed and updated independently. The interaction between each layer enables real-time distributed message processing and distributed management of multi-service processing in a city-main based, high-speed distributed environment. By accelerating message processing, using a dedicated buffer pool created for each application process, and monitoring the status of all messages in the buffer, the high reliability inherent in dedicated platforms is ensured. In addition, it minimizes the waste of hardware resources by running application processes as daemons in buffer pools. In particular, to provide domain-based multi-service, it supports RMPF grouping by task to distribute tasks to appropriate domains to eliminate system bottlenecks. It is also implemented in a modular way to achieve both scalability and adaptability through various multiservice distributed management schemes such as multiservice API, recovery, monitoring, and multiservice provisioning.

INDEX TERMS Multi service, distributed message processing, unified platform, memory management, multi domain.

I. INTRODUCTION

The Fourth Industrial Revolution has spurred remarkable advances in information and communication technology (ICT), particularly with the proliferation of smart devices. This surge has led to an exponential increase in real-time data generation, especially in areas such as personal card purchases, transport card authorizations and smart factory operations [1], [2]. As a result, there is a growing demand for high-speed, real-time message processing platforms that are essential for effective data management and processing [3].

High-reliability domains such as financial payments and IoT-based smart factory systems require platforms that provide fault tolerance, independent memory management and real-time data processing. Until now, only dedicated platforms such as Tandem, Stratus, SUN and Sequoia have met these stringent requirements, providing continuous monitoring, fast fault recovery and exceptional system reliability [4], [5]. However, these platforms are prohibitively expensive to implement and maintain. However, because these platforms

The associate editor coordinating the review of this manuscript and approving it for publication was Fabrizio Messina.

are developed and implemented with expensive, dedicated hardware and software as an integrated whole, they are very expensive to operate and maintain, and their complex structure makes them expensive and difficult to scale. In addition, general purpose platforms, while more economical, present several challenges. Because general purpose platforms adopt general purpose software on general purpose hardware, they have low cost and high compatibility, but they have the disadvantages of limited performance and poor system stability and reliability. In addition, general purpose platforms have difficulty handling real-time multiservice processing, mainly due to their message processing design. They also lack efficient distributed message processing, which often results in excessive consumption of hardware resources [6], [7]. In addition, platforms such as Apache Tomcat and Tmax TP Monitor often suffer from poor monitoring and recovery capabilities, with limited or no native monitoring capabilities. External application performance management solutions often fill these gaps, but they are additional integrations and not inherent to the platform [8].

In this paper, we present the Real-time Message Processing Framework for Multi-Business Domains (RMPF), an innovative integrated platform that bridges these gaps and provides real-time distributed message processing for multi-service domains. The reason for choosing the RMPF architecture in this paper is to use the hardware of a general-purpose platform to reduce the price competitiveness and implement the same features and performance as a dedicated platform in software. In addition, the proposed platform is designed to secure high performance and reliability with low cost, and it is modularized by function, so it is easy to add other general-purpose systems and is highly scalable.

RMPF bridges these gaps by combining the high reliability and system efficiency of a dedicated platform with the economic benefits of a general-purpose platform. It is also based on a general-purpose platform but implements the functionality of a dedicated platform to handle faults, errors, and exceptions for high reliability and competent system management, which ensures message processing [9]. In particular, it uses a dedicated message buffer pool for real-time multi-service application processing, efficient distributed message processing and optimized hardware resource utilization. It also incorporates domain-based high-speed distributed processing, a modular design for scalability, and a suite of management functions, making it versatile for both cloud and legacy systems [10].

The proposed platform manages resources within its own platform, which allows for efficient utilization of resources and reliable management. We also compared it to Apache Tomcat, a widely used general-purpose platform, in a rigorous performance evaluation, and the results were quite good. In all four tests, our platform consistently showed a significant reduction in CPU usage. A key factor in this efficiency is the platform's architecture, which uses distributed message processing with a message buffer pool and maintains processes as daemons.

In summary, our main contributions can be summarized as follows:

1) We present an integrated platform, RMPF, which provides real-time distributed message processing for multi-service domains.
2) It supports multilayer architecture which achieves extensibility and flexibility by providing modularization in each layer which can be designed and updated independently.
3) It provides the capability for fast message processing while achieving reliability by using the dedicated buffer created exclusively for each application process residing as a daemon for the incoming message, and then tracking the status of all messages in the buffer through monitor.
4) When messages come into the platform, the message buffer pool is used to ensure message delivery. Messages that enter the platform are stored in the message buffer pool, and messages are classified and tracked through page-by-page indexing and unique hash value generation. Processed messages are sent internally or externally depending on their destination, and the entire process is monitored in real time by the monitoring module to enable rapid recovery in the event of a failure.
5) It minimizes the waste of hardware resources due to memory fragmentation and frequent process creation arising in a general-purpose platform by keeping the application process with its own memory space in the buffer pool.
6) It provides domain-based multi-services by supporting the grouping of RMPFs according to domains, and remove the system bottleneck by enabling the distribution of tasks to the proper domain.
7) It is further enhanced by various multi-service distributed management schemes such as multiservice API, recovery, monitor and multiservice deployment, each of which is implemented in a modular fashion that ensures both scalability and adaptability, making it suitable for a wide range of environments from traditional legacy systems to modern cloud infrastructures.

This paper is organized as follows: in Section II, we review several memory management techniques, recovery and distributed service processing middleware as related works. In Section III, we describe the architecture of our proposed platform, and in Section IV, the overall operation of the system. In Section V, we evaluate the performance of our proposed system, and finally, in Section VI, we give a conclusion.

II. RELATED WORKS

This chapter introduces existing methods for memory management techniques, recovery techniques and distributed service processing middleware, which are critical components of the platform, and discusses their limitations. It also presents proposed methods to address the limitations of existing methods.

A. MEMORY MANAGEMENT TECHNIQUES

Memory management is very important in computer systems to ensure efficient utilization of memory resources and to prevent errors caused by memory misuse. In this chapter, we shall describe about common types of memory management techniques.

1) CONTINUOUS MEMORY ALLOCATION TECHNIQUES

Contiguous memory allocation is one of the simplest forms of memory management, allocating a single, uninterrupted block of memory to each process. It's simple, fast and was widely used in early computer systems. The main disadvantage of this technique is fragmentation, where memory is wasted as unusable white space. As processes are loaded and unloaded from memory, available blocks of memory of different sizes may be scattered around, leaving too little space for new processes, and even if the total free memory is sufficient, it may be fragmented into small chunks that cannot be used by larger processes. Also, if a process does not fully use its allocated block of memory, internal fragmentation occurs, wasting the unused portion. When memory is fragmented, it can slow down the system and introduce memory allocation overhead. Preventing this external fragmentation may require memory compression, which can be time consuming and degrade system performance [11]. In addition, as the number of processes and memory requirements increase, contiguous memory allocation becomes less feasible and more inefficient, limiting scalability. In addition, as processes grow in size, there is not enough space to store them in the allocated block, requiring processes to be moved, which is inefficient [12]. Recognizing these drawbacks, many modern operating systems are moving towards more flexible memory management techniques such as paging and partitioning. In this paper, we propose a solution to the fragmentation and slowdown problems in these techniques, i.e. by storing incoming messages in a message buffer pool with their hash value instead of in a message queue, to support no memory movement by processing by reference. This solves the problem of memory fragmentation and slowdown.

2) PAGING TECHNIQUES

Memory is divided into fixed-size blocks called pages, and each process consists of fixed-size blocks. A process's pages can be distributed throughout physical memory. This reduces fragmentation, since smaller fixed-size blocks fit better than larger variable-size blocks. Each page is mapped to a frame in physical memory, and a page table keeps track of where each page is in physical memory. Its advantage is that it makes it easier to swap pages in and out of RAM to accommodate more processes, while its disadvantages are that it takes time and resources to maintain and retrieve the page table, which creates overhead; it also wastes space, since the last page of a process is not fully utilized, which can lead to internal fragmentation; the page table size can be large if the process is large, which can cause the page table to grow and require more memory. Therefore, there is the need for multiple memory accesses (double references). One for the page table and one for the actual data. This makes accesses slow, and frequent page swaps between RAM and disk cause thrashing that can seriously degrade performance. Paging mitigates the fragmentation caused by contiguous allocation, but it also introduces its own problems [13], [14]. To address these issues, the proposed platform creates, and uses message buffer pool to minimize memory usage, thus eliminating paging and minimizing throughput and performance degradation.

3) PARTITIONING TECHNIQUES

Partitioning is one of the primary memory management techniques used, especially in early computer systems, and involves dividing physical memory into multiple partitions to accommodate multiple processes. Main memory is divided into multiple partitions. Each partition can hold exactly one process. When a process enters main memory, it is assigned a partition. The size of a partition can be fixed or variable, and memory is divided into segments based on different types of data or instructions. Each segment can be expanded or shrunk independently. Partitioning allows you to divide a process into logical segments, such as code, data, stack, and so on. Each segment has a base address and a limit, and the system protects against unauthorized access by ensuring that the accessed address is within the segment limit [15], [16]. This technique is used when the message size is small and is not suitable for processing large amounts of message data.

4) VIRTUAL MEMORY TECHNIQUES

Virtual memory technology allows processes to run even when they are only partially located in primary memory. The rest of the process resides in secondary storage, which is usually a hard drive. Virtual memory uses both hardware and software to allow a computer to use disk space to compensate for a lack of physical memory. When a part of a program or data that is not in main memory is accessed, a page fault occurs [17]. The operating system then replaces the required page with memory, and replaces other pages if necessary. This gives the user the illusion that the available memory is almost unlimited, and also introduces potential performance degradation due to overhead swapping due to context switching between main memory and disk [18], [19]. To address these issues, the proposed platform creates a message buffer pool to minimize memory usage, and minimize processing speed and performance degradation.

5) GARBAGE COLLECTION TECHNIQUES

Garbage collection is primarily used in the context of advanced programming languages to automatically reclaim memory that is no longer in use or referenced by the program. Languages like Java and C# have built-in garbage collectors, and these systems periodically check for data in memory that is no longer accessible to the program. When such data is found, the memory it occupies is freed and returned to the system for other uses. While this technique reduces the risk of memory leaks, it incurs an overhead due to the garbage collection process [20]. To address this overhead, the proposed platform uses a message buffer pool, which means that the memory is standardized to a certain size to minimize memory fragmentation and overhead.

6) DATA PAGE BUFFER POOL TECHNIQUES

This technique is one of the methods for managing data stored in memory for processing data acquired through a network device. As a method of processing packet data acquired through a network device, the packet data acquired through the network device is delivered to a central processing unit (CPU) or written directly to memory at the request of the CPU so that the central processing unit can process the packet data. In general, memory provides a relatively compact and quickly accessible memory structure. However, while this technique is suitable for small-scale and limited memory management, it suffers from system inefficiency when applied to large-scale memory management due to the frequent increase in processing count and the increase in management points [21]. The platform proposed in this paper enables large-scale memory management through large-scale message processing by organizing and utilizing a message buffer pool in the memory in addition to small-scale.

7) LOOKUP TABLES TECHNIQUES

This technique is a memory allocation technique using a lookup table, which converts the requested memory size into a key value of logarithmic size to access the lookup table, which is a hash table, and searches for a free memory block of the requested size from the linked list of blocks with the corresponding logarithmic size. In addition, the free memory blocks are organized as a double linked list with a FIFO policy, and all memory blocks are managed as an addressed ordered double linked list, which has the advantage of performing memory merges quickly when returning memory. However, as the memory size increases, the number of processor operations increases due to the memory fragmentation phenomenon, resulting in a slowdown and decrease in system efficiency [22], [23]. To solve this memory fragmentation phenomenon and decrease in system efficiency, the proposed platform standardizes the buffer memory to a fixed size in order to minimize memory fragmentation, and hence prevent slowdown and system degradation.

B. RECOVERY

The primary goal of a database system recovery mechanism is to ensure that the database can be restored to a consistent and correct state after any kind of failure, such as a system crash or media failure. Recovery mechanisms ensure the ACID properties (atomicity, consistency, isolation and durability) of a database system. The main focus of this chapter is on popular recovery mechanisms, specifically log-based recovery techniques and shadow paging recovery techniques [24], [25].

1) LOG-BASED RECOVERY TECHNIQUES

Log-based recovery techniques use logs to record all operations on the database. This log helps to return the system to a consistent state after a failure. The basic principle of log-based recovery is that the actions of each transaction are recorded in a log before they are applied to the database, ensuring that these records can be used for recovery purposes [26].

- *Immediate Update Technique:* In the immediate update technique, changes are applied directly to the database even before the transaction is committed, but they are also written to the log. Complexity is increased by the need to support both undo and redo operations. More I/O operations compared to lazy updates because the changes are applied immediately. Must deal with transactions that may have been partially completed during a crash [27].
- *Deferred Update Technique:* In a lazy update technique, changes are logged but not immediately applied to the database. Changes are only applied after the transaction is committed, potentially increasing recovery time as all transactions may need to be re-executed since the last checkpoint. It is necessary to keep a significant portion of the login in memory until the transaction is committed [28].
- *Check Point Recovery Techniques:* Checkpoint recovery techniques take regular snapshots of the system to minimize the number of log entries that need to be considered during recovery. The disadvantage is the trade-off between recovery time and system overhead; frequent checkpoints can reduce recovery time, but at the expense of system performance. Also, if not implemented carefully, concurrent transactions during checkpoints can complicate recovery.

A common disadvantage of log-based recovery techniques is storage overhead, as logs can grow over time and consume significant storage. There is also a performance overhead, as writing to logs and databases can double I/O operations, especially on highly transactional systems. Managing, archiving, and deleting old log records can be complex, and there are concurrency issues, requiring proper synchronization, especially when multiple transactions access the logs in a distributed system. In addition, a combination of logs and periodic backups is required for a complete recovery mechanism, and relying on logs alone may not be sufficient in the event of a catastrophic failure. In practice, despite these drawbacks, log-based recovery remains one of the most reliable ways to ensure database consistency and durability, especially when combined with other strategies such as backup and replication [29].

2) SHADOW PAGING RECOVERY TECHNIQUES

Shadow paging is an alternative to the log-based recovery model; instead of using logs, this technology uses a paging mechanism. The process is that when a transaction modifies a page in the database, instead of overwriting the original page, a new "shadow" page is created to record the changes. The original remains unchanged, and when the transaction commits, the database pointer is updated to point to the new shadow page, which becomes the current page [30]. The advantage of this technique is immediate recovery. In case of failure, the system can revert to the previous page, and no undo or redo logs are required as the previous page naturally acts as a backup. On the downside, shadow paging can lead to fragmentation because changes create new shadow pages. It can also require more storage space as the original and shadow pages coexist for a period of time [31], [32].

To solve the shortcomings of the log-based recovery and shadow paging recovery methods presented above, the proposed platform proposes a message swap method in the message recovery function. This method replicates data from the message buffer pool and stores the same data on disk as an archive. In addition, this technique recovers message failures by using the message buffer pool in case of failure.

C. DISTRIBUTED SERVICE PROCESSING MIDDLEWARE

In the era of distributed systems and cloud computing, middleware plays an important role in ensuring seamless communication, data exchange, and process management across different services. In addition, multi-protocol gateway solutions that facilitate data exchange between entities with different technical origins provide efficient data exchange in the gateway context for major automotive IoT Ethernet-based communication technologies in a variety of service-oriented architectures, requiring compatibility and real-time response capabilities [33], [34].

Distributed service processing middleware also plays an important role in facilitating and managing interactions between different services and components so that they can operate consistently in a distributed environment. Middleware abstracts the details of where a component or service is running [35], [36], meaning that clients do not need to know a specific location within the distributed system when trying to access a service. Middleware also provides mechanisms to handle concurrent access, ensuring data consistency and integrity [37]. And middleware provides mechanisms for failure detection, failure masking and recovery. On the other hand, middleware provides various functions such as load balancing, security protocols and service discovery to efficiently, and effectively manages operational issues in a distributed environment [38], [39].

1) APACHE TOMCAT

Apache Tomcat is a popular open source middleware solution that acts as a Java servlet container. It also creates threads or uses other mechanisms to parallelize or execute tasks in parallel. However, if you have specific requirements for external processes, Java provides a way to launch and manage them. On the other hand, the creation process can consume a lot of system resources. Especially in a web server environment, where many requests can be processed simultaneously, these resources need to be managed appropriately. Also, this method does not provide error handling, so you will need to work separately on comprehensive error handling. In addition, since there is no process management, it can sometimes become a "zombie" process, and it is tedious to clean up the resources after the process has finished [8], [40]. To address these issues, the proposed platform uses Message Router (MR) units to enable large-scale concurrent processing, and a message buffer pool is used for error handling and process management via the monitor function.

2) OBJECT REQUEST BROKER (ORB)

An ORB is a middleware system that enables communication in distributed object-oriented systems by allowing program calls to be made from one computer to another over a network. Through location transparency, the calling application does not need to know the location of the remote object. ORBs also rely on interface definitions such as IDL to define and expose object methods. Advantages include the ability to seamlessly integrate and communicate object-oriented systems over networks, and to provide a high level of abstraction and encapsulation. Disadvantages include potential performance overheads due to the abstraction layer and the complexity of setting up and managing distributed objects [41], [42].

3) TRANSACTION PROCESSING MONITOR (TPM)

A TPM is a middleware system designed to manage and monitor the execution of a series of tasks, ensuring that these tasks are completed as a single unit (transaction). In essence, it ensures that all tasks in a transaction are completed and that nothing is completed if they are not. It also ensures that each transaction runs independently of the others. Pros: It ensures the ACID properties of transactions in distributed systems: atomicity, consistency, isolation, and durability, and provides robustness and reliability for applications that require transactional integrity. Disadvantages include latency due to transaction confirmation and locking, and potential race conditions that can occur when many transactions compete for the same resources [43].

4) MESSAGE-ORIENTED MIDDLEWARE (MOM)

Message-oriented middleware is a category of middleware that facilitates communication between distributed components through message passing. Components communicate by sending and receiving messages, even if they are not online at the same time. Because components communicate asynchronously, they do not have to wait for a response after sending a message, and if the receiving component is unavailable, the message is held in a queue until it is delivered. The advantages are that components can operate

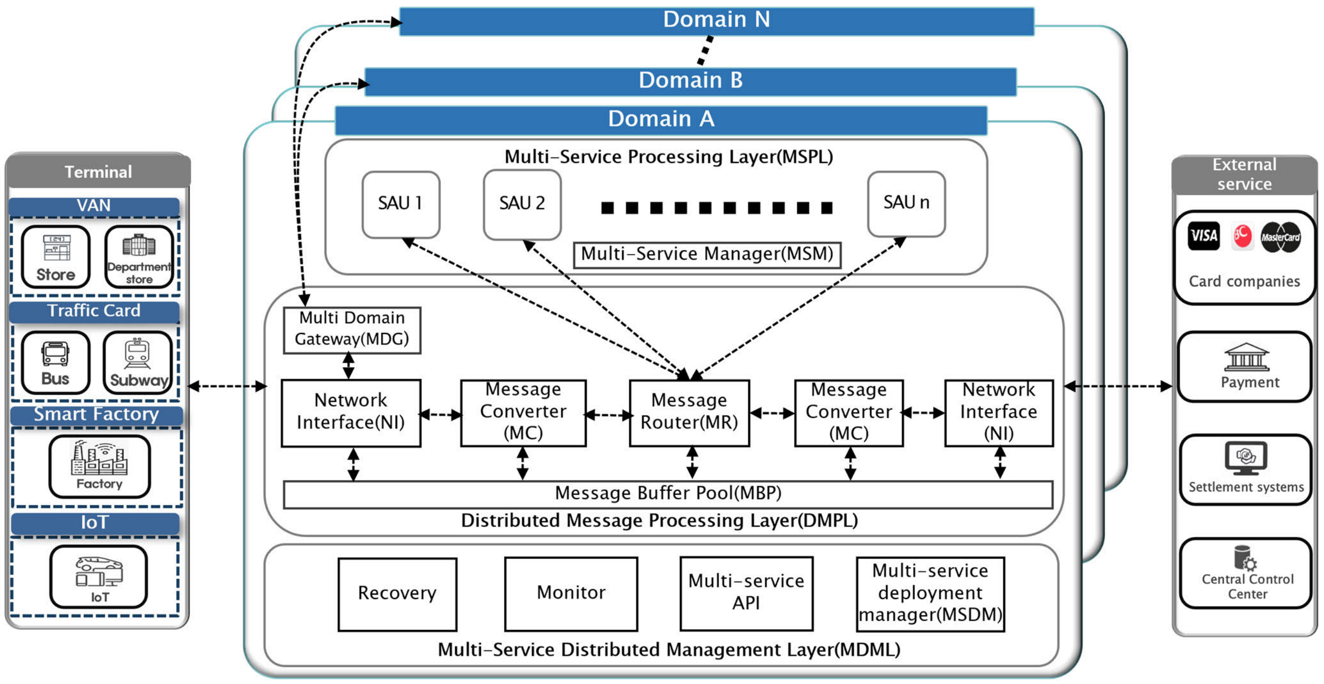

FIGURE 1. RMPF architecture.

independently, making them highly scalable and resilient to component failures due to message queues. The disadvantages are that message delivery is not always guaranteed and the asynchronous nature can complicate application logic and error handling.

To improve these problems, a message queuing approach was used in [9]. However, the RMPF proposed in this paper improves the problems in [9], which greatly improves the message processing speed and system efficiency of the platform. In addition, the proposed platform separates units into independent layers, which simplifies the connection steps, reduces the number of processing steps, and greatly improves the system efficiency.

Furthermore, since the existing system uses OS-managed queues, queue congestion slows down the overall operation, while our system uses dedicated buffers from a pool of message buffers created exclusively for each application process, which speeds up message processing.

III. RMPF ARCHITECTURE

RMPF is designed to use the hardware of a general-purpose platform to achieve the functionality and performance of a dedicated platform at a lower price point. In addition, it is designed to secure excellent performance and high reliability despite its low price, and it is easy to add other general-purpose systems and is highly scalable because it is modularized by function. In addition, it manages resources within its own platform, enabling efficient utilization of resources and reliable management. In addition, RMPF is an integrated platform that provides real-time distributed message processing to various terminals such as payment companies, banks, settlement systems, and central control centres, and multi-service domains such as VAN, transportation cards, smart factories, and IOT. As shown in Fig. 1, it consists of three layers: Distributed Message Processing Layer (DMPL), Multi-Service Processing Layer (MSPL) and Multiservice Distributed Management Layer (MDML). RMPF supports multilayer architecture which achieves extensibility and flexibility by providing modularization in each layer which can be designed and updated independently. The interaction between those layers enables multiservice processing using real-time distributed message processing and multiservice distributed management on domain-based high-speed distributed environment.

A. DISTRIBUTED MESSAGE PROCESSING LAYER (DMPL)

DMPL receives each message from terminal, then sends it to the corresponding service application unit (SAU) in MSPL, which in turn sends the additional processing request to the external service via DMPL according to the processing results from SAU. It consists of five components: Network Interface (NI), Message Converter (MC), Message Router (MR), Message Buffer Pool (MBP) and Multi Domain Gateway (MDG) as shown in Fig. 2. NI receives a message from terminal, which in turn is converted into application format by MC, and then sent to the proper application process in the corresponding SAU by MR. Each SAU consists of several application processes for the same application. MC converts external messages to internal common data format for SAU, providing easy maintenance and extensibility for SAU According to

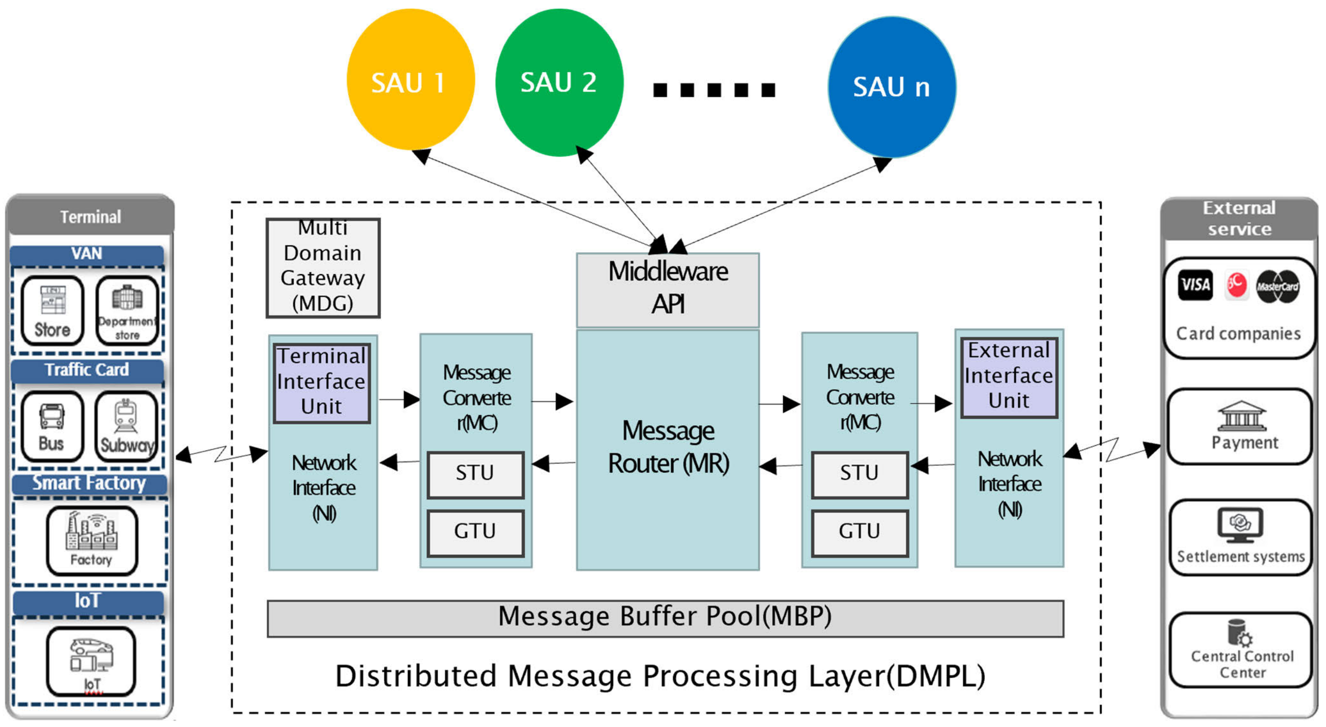

FIGURE 2. **DMPL configuration.**

the processing result of the SAU, MR sends the additional request to the external service via MC and NI to further process external services such as card company service from stores, payment settlement system from bus, smart automation system from factory and central control center from IoT sensors. MBP stores the messages from NI, MC and MR for buffering, and manages a queue for each process in SAU for easy migration of task between processes in the same SAU. Each RMPF has its own unique domain to which it belongs to, and each message stores information about thedomain of RMPF on which it can be processed. MDG has the role of sending the message received from NI to the proper RMPF in the corresponding domain for the message.

1) NETWORK INTERFACE (NI)

NI handles messages from various external terminals, and connects to the external services. For example, messages from stores are connected to card company service, from bus to payment settlement system, from factory to smart automation system, from IoT to central control center, etc. NI has consisted of two components: terminal interface unit and external interface unit. Terminal interface unit receives message, and checks, for each message, its domain. If it belongs to the other domain, it forwards the message to MDG so that it can be processed in the proper domain; otherwise sends it to MC. External interface unit connects to the proper external service according to the processing result from SAU in MSP, receives the result from the external service, and sent back to the SAU, which in turn completes the task by returning the final result to the terminal through terminal interface unit. External interface unit acts as bridges that provide connectivity between RMPF and external service, and hence play a pivotal role in integrating internal subsystems with external organizations, ensuring that data can move flexibly and efficiently across various boundaries.

2) MULTI DOMAIN GATEWAY (MDG)

RMPFs can be divided into multiple business domains or area domains. MDG supports the grouping of RMPFs according to domains by providing a link between domains in multi-domain construction, and remove bottlenecks by enabling the transfer of tasks to the proper domain. MDG in the current RMPF performs the role of calling the task in the master RMPF of residing in the domain for the incoming message, which in turn processes it, returns the result to the current RMPF unless the bottleneck occurs; otherwise performs the load balancing by transferring the task to the other RMPF in the same domain according to its priority order.

3) MESSAGE BUFFER POOL (MBP)

MBP is essentially a dedicated memory space or storage designed to temporarily store incoming intermediate messages before they are processed. Its primary role is to act as a cushion for sporadic or unexpected spikes of incoming data, ensuring that the system is not overloaded, and each message is processed efficiently in real time. By providing a buffer for incoming data surges, intelligently categorizing messages, andensuring rapid message processing, MBPs

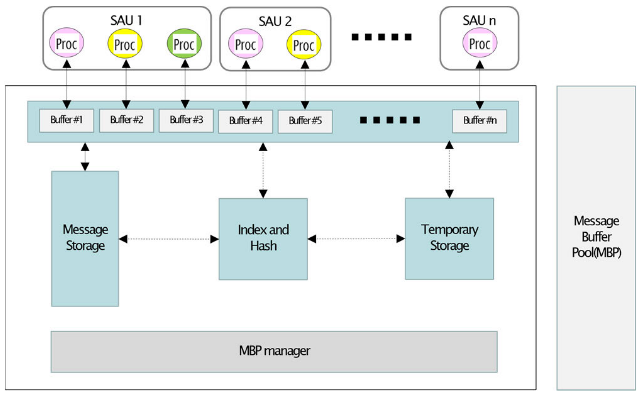

FIGURE 3. MBP components.

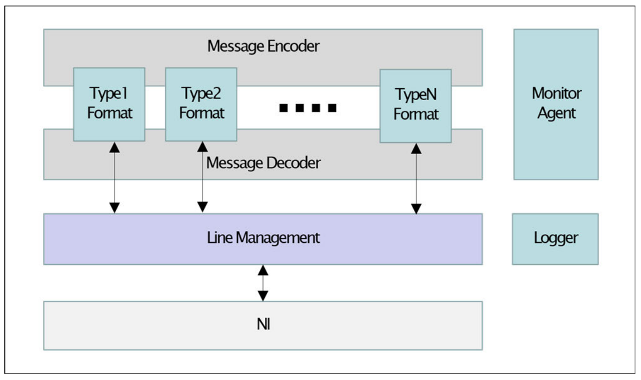

FIGURE 4. STU configuration.

greatly improve the efficiency, responsiveness and reliability of such systems. From a design perspective, MBP is geared toward rapid allocation and deallocation of buffers. This means that as soon as a message is received, it can be quickly placed in a buffer, and once processed, the buffer can be immediately freed up for the next message. This approach minimizes latency and ensures smooth data flow through the system.

As in Fig. 3, it consists of five components: message storage, index and hash storage, buffer, temporary storage and MBP manager. Message storage stores intermediate messages, and index and hash storage stores information for accessing messages. Buffer is a queue for storing index and hash information for the messages of each process in SAU, and temporary storage stores the index and hash information removed from the buffer when the bottleneck in buffer space occurs.

The existing system makes use of the queue managed by OS, thus causing the slowdown of the overall operation and reliability degradation due to the overload arising in the queue, while our RMPF speeds up the message processing, and achieves reliability by using the dedicated buffer in MBP exclusively created for each application process in SAU, and following up the status of all the messages in the buffer in MBP through monitor in MDML. In the existing system, as the memory size of buffer varies, the number of operations in each process of SAU increases due to the memory fragmentation resulting in slowdown and decrease in system efficiency.

RMPF provides an efficient MBP management scheme for the dedicated buffer memory space which can minimize memory fragmentation, and hence prevent system degradation. To solve this memory fragmentation phenomenon and decrease in system efficiency, RMPF standardizes the buffer memory to a fixed size, and transfers the content of the buffer into temporary storage in case overload occurs in the buffer in order to minimize memory fragmentation, and hence prevent slowdown and system degradation. Moreover, we store the hash and index values of incoming messages into buffer instead of the message data in order to support data processing by reference without memory movement, preventing the problem of slowdown.

MBP manager provides a buffer pool management scheme for generating a plurality of message storage organized in parallel, generating index and hash storage for storing index and hash information for the messages in message storage in parallel, generating buffers each with fixed size for storing index and hash information for the messages into each process of SAU, and generating temporary storage for storing the data removed from buffer when the bottleneck event is determined to have occurred. Thus, it is possible to omit the process of additional memory allocation by creating a buffer pool in advance, and to allow the application process to directly access the previously created buffer pool thereby reducing the burden of overloading caused by the process for memory allocation.

Temporary storage capability is especially important for handling high-throughput scenarios or ensuring that messages are not lost or delayed due to processing bottlenecks during peak load times.

In addition, in a multi-business domain framework, messages often have varying priorities and complexity. MBPs can be equipped with intelligent mechanisms which not only store messages, but also classify or prioritize them. For example, in a system that processes both financial transactions and user feedback, financial messages may be given a higher priority so that they are processed before feedback messages. This prioritization ensures that business-critical or time-sensitive data is processed immediately.

4) MESSAGE CONVERTER (MC)

MC converts messages from a terminal format to an application format and vice versa. It consists of two types of units: Single Type Unit (STU) converts from single terminal format to the corresponding application format, and vice versa. A Group Type Unit (GTU) converts a group of terminal formats to the corresponding application format, and vice versa. MC executes parsing, encoding and decoding different external messages. It also uses ISO 8583 and Free Format as internal message standard that is common within

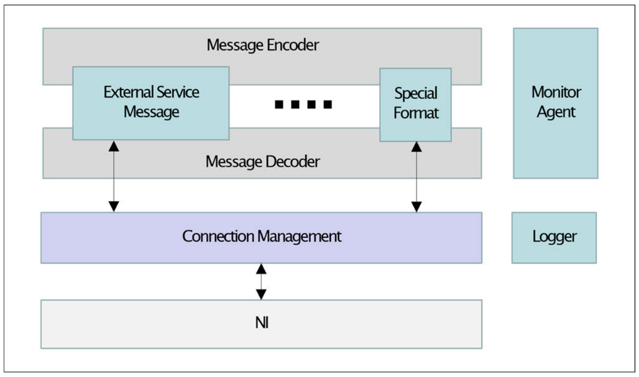

FIGURE 5. GTU configuration.

the platform. The input process of MC receives messages from the network interface, converts them to the appropriate application format, and transfer them to MRU. The output process of MC receives messages from the MRU, converts them to the appropriate terminal format, and transfer them to NI, which intern sends them to the appropriate external service.

- *Single Type Unit (STU):* STU is responsible for analyzing, converting serially incoming messages into proper single type from terminal interface unit of NI to MR, or vice versa. It consists of connection management, parser, encoder and decoder, logger and monitor agent as shown in Fig. 4. Connection management module manages the status of terminal equipment, and provides flexibility to cope with new equipment as it becomes available without making major system-wide changes when you add new types of terminals or new communication methods. Messages are parsed, encoded and decoded in a prescribed manner, and all messages are logged in real time as needed. The timer is activated after the text message is received, and if the result is not received within the specified time, it will automatically fail. Monitor agent checks the status for each converted message in order to maintain the correct format of the message.
- *Group Type Unit (GTU):* GTU converts a group of incoming messages from MR into a complex message, and sends it to external interface unit of NI or versa. It can greatly decrease communication overhead by encapsulating a large volume of messages to the same external service into one group message and processing them in a batch at once. As shown in Fig. 5, it consists of connection management, parser, encoder and decoder, logger and monitor agent. Similarly, as STU, connection management module manages the status of terminal equipment, and message encoder and decoder receives a group of messages from MR, convert it to a complex message or vice versa, and all messages are logged in real time as needed. The timer is activated after the text message is received, and if the result is not received within the specified time, it will automatically fail. Monitor agent checks the status for each converted message in order to maintain the correct format of the message.

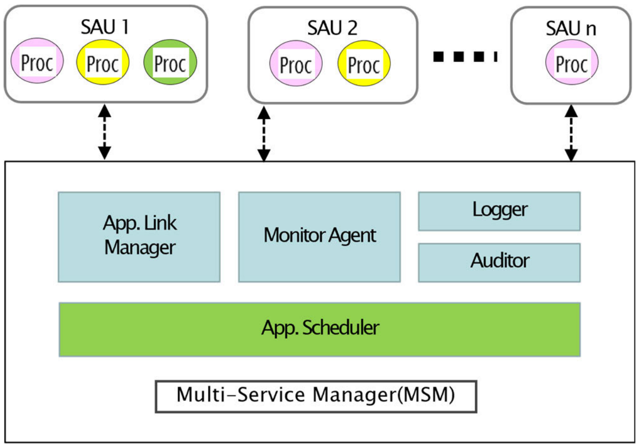

FIGURE 6. MSM components.

5) MESSAGE ROUTER (MR)

MR receives messages from the MC, routes them to the MSP's Message Service Manager (MSM) for processing, and, if necessary, sends the results from SAU to the MC to access external services through the NI, or returns the final termination status to terminal via MC and NI.

B. MULTI SERVICE PROCESSING LAYER (MSPL)

In the realm of real-time message processing for multi-business domains, the multi-service processing module is emerging as a key element. It enables the efficient and seamless integration of various business processes each ofwhich process messages from different sources simultaneously. It consists of Multi-Service Manager (MSM) and Service Application Unit (SAU).

1) MULTI SERVICE MANAGER (MSM)

MSM plays an important role in orchestrating the various services in a real-time message processing framework. As shown in Fig. 6, it consists of application link manager, application scheduler, monitor agent, logger and auditor. It acts as a central hub to coordinate message routing to the proper application service process in SAU through application link manager, and ensure proper service invocation by application scheduler. MSM also prioritizes tasks, handles service failures, transmits various statuses, message processing information, logs, and protects the integrity of message transactions in a multi-business environment by monitor agent, logger and auditor. In addition, it enables loss-less processing of messages through application-specific queue management for each SAU even in the event of overload, solving the user's risk of message loss.

2) SERVICE APPLICATION UNIT (SAU)

SAU is a dedicated processing component designed to execute several identical processes for the specific application service within the framework. That is, in each SAU, the distinct business service is covered, and the clones of the same process that performs the same task are managed as a group. For example, SAU 1 is for banking logic, SAU 2 for card company logic, and so on.

Essentially, each SAU ensures that each service request is met in a customized execution environment, optimized for performance, and suited to the unique requirements of that service. Therefore, each SAU can be customized for different service type. The use of SAUs also greatly simplifies the handling of multiple services. Each service can be queued, run, and managed in the corresponding SAU with a level of precision that is difficult for traditional systems to achieve. SAU is also the core of the multi-service processing module, responsible for real-time data processing and multi-service on the platform with adaptive design, resource efficiency and seamless integration.

By isolating each service application and providing a customized environment, the SAU ensures optimal efficiency and performance for each task. It is also a pivotal component of modern distributed systems designed to encapsulate specific service logic and functionality. Operating as modular units within a larger system, SAUs streamline the execution of specific tasks to ensure both efficiency and accuracy.

C. MULTI-SERVICE DISTRIBUTED MANAGEMENT LAYER (MDML)

In the realm of distributed systems, managing multiple services effectively and efficiently is paramount. MDML addresses this complexity, and provides various management schemes for recovery, monitoring, multi-service API and multi-service deployment manager to facilitate service coordination, deployment and fault tolerance.

1) RECOVERY

Recovery is critical in a multi-service distributed management module. In a distributed environment, services can fail for a variety of reasons, including network issues, hardware failures, software glitches and unexpected traffic spikes. Effective recovery mechanisms are therefore essential not only to ensure service continuity, but also to ensure data integrity and maintain user trust. Backup and restore mechanisms ensure that service state and data are regularly backed up so that services can be restored to their most recent and consistent state in the event of a failure. Transaction logging is also necessary in distributed environments where multiple services often interact in complex ways. Transaction logs provide a detailed record of all these interactions. In the event of an error, these logs are critical to understanding the sequence of events leading up to the error and ensuring that all services can be rolled back or forwarded to a consistent state.

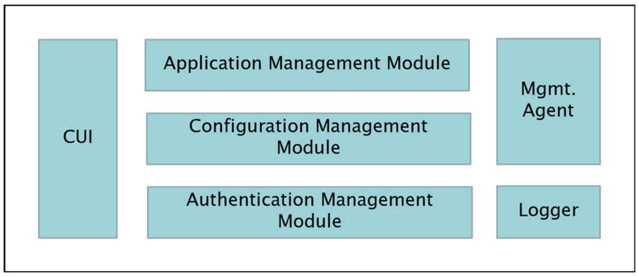
CUI
Application Management Module
Configuration Management Module
Authentication Management Module
Mgmt. Agent
Logger

FIGURE 7. Monitor components.

The proposed platform supports message loss prevention during failover and upgrades through hot swap support. The hot swap feature saves data to disk as an archive as it comes into the message buffer pool, and then watches each module and unit on the monitor, and monitors the processing status of each message. In the event of a failure, the message buffer pool data is swapped with the data stored on the disk for recovery. In addition, the following processing is done to ensure message processing because it can solve the risk of message loss when sending messages. In the case of faults, if the message from the outside is not recognized, it is retransmitted according to the set rules by repeating the processing 3 times, and if an internal fault occurs, the process (SAU) is stopped and the processing is handed over to another process (SAU) waiting for processing. In addition, if an error occurs and the fault cannot be handled and is outside the scope of exception processing, the error is checked in the status monitoring and the error message is alarmed through dump processing so that the administrator can check it. In addition, it does not affect the system as a whole when handling the error. And in the case of exceptions exception handling, the message is exceptionally handled by creating a certain event rule corresponding to the exception. With these features, the proposed RMPF can be used in various fields such as finance, transport, IoT, and retail.

2) MONITOR

Monitor continuously monitors the status of all services, and watches the flow and processing of all the messages within the platform. This proactive approach allows for early detection of potential problems before they escalate into larger issues. Therefore, it achieves reliability by creating the dedicated buffer in MBP exclusively for each application process, and following up the status of all the messages in the buffer in MBP through monitor.

As shown in Fig. 7, it consists of application management, configuration management, authentication management, command line user interface (CUI), management agent and Logger. Application management module starts, stops, and monitors all the online and batch application processes. Configuration management module stores all the information for the configuration of each component in RMPF, and Logger collects the status of each component in real time. Authentication management module manages, and controls

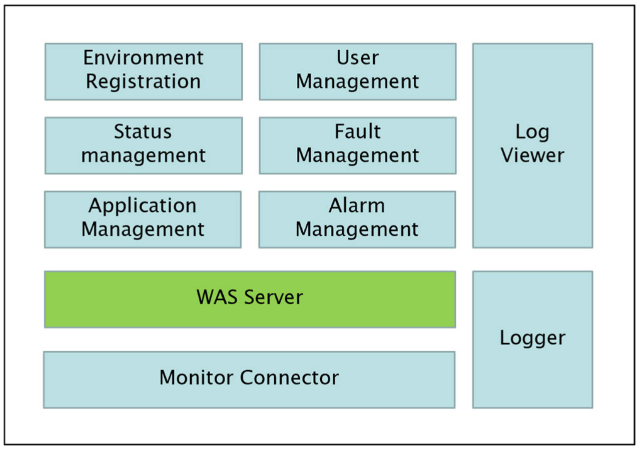

FIGURE 8. MSDM configuration.

user permissions to access the framework, and CUI provides emergency failure handling functions. Management agent transmits the information of each component in RMPF to MSDM.

3) MULTI SERVICE API

In today's interconnected digital ecosystem, the ability to interface, interoperate, and integrate diverse services is more important than ever. The rise of cloud computing, microservices architectures, and IoT has led to a proliferation of different services, each offering unique capabilities. Multiservice application programming interfaces (APIs) have become key in this scenario, ensuring seamless integration, scalability and accessibility of numerous services. A multiservice API is an interface that facilitates communication and data exchange between many different services, regardless of their underlying architecture or platform. Rather than a single-purpose API dedicated to a specific service, a multiservice API serves as an integration layer that binds various service endpoints into a unified and accessible interface. It also provides framework functions for efficiency and structuring such as business development, and standard functions for use in business. It supports various languages such as NMC and C, and provides a dynamic framework with built-in major functions for creating applications. It also provides functions for communication with the modules of the framework and libraries in the form of packages that facilitate the construction of various business systems.

4) MULTI SERVICE DEPLOYMENT MANAGER (MSDM)

MSDM provides customized tools and features to supervise, control and optimize the operation of multiple individual services. The proposed framework provides an EMS in a separate web environment, providing a UI for administrators to remotely manage the main functions of the approval system. The main functions include environment registration, user management, status management, fault management, application management, alarm management, and log inquiry, and communicate with the Monitor Unit in real time through the Monitor Connector Module to collect and provide the latest information. Fig. 8 shows the configuration of MSDM, and each component plays the following roles.

Environment Registration serves as the entry point for integrating new environments or systems into the monitoring framework. Administrators specify, and configure the parameters of each environment so that monitoring tools can recognize all assets, and collect relevant data correctly. User Management manages user profiles, roles and permissions. It also determines who can access the monitoring framework, what they can see, and what actions they can take. Status Management is an essential component that provides real-time status information for all monitored assets, capturing the operational status of services, applications and environments. Fault Management continuously scans the system for anomalies or failures. Application Management oversees the deployment, updating, monitoring and retirement of applications within the monitored environment. Alarm Management raises alarms based on predefined conditions or thresholds. Log Viewer provides a user-friendly interface to view, search, filter, and analyze logs generated by the platform. Web application server (WAS) hosts the web-based interface of the monitoring framework, allowing users to access the tools through a browser. Logger is responsible for recording events, transactions, errors and other important activities within the platform. Monitor connector provides a passage for connecting to the monitor.

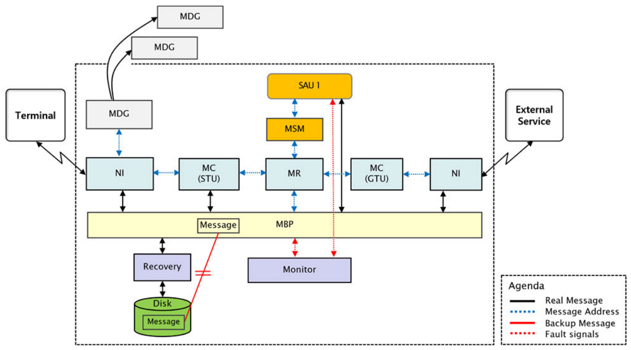

FIGURE 9. Framework message processing diagram.

IV. DATA PROCESSING FLOW

The data input to the Network Interface consists of various messages such as card authorization/cancellation messages from various external terminals, transportation cards, smart factory data and IoT. When a message is entered, NI checks the corresponding domain through the multi domain Checker. If the system is divided into multiple business domains, the message is routed to the appropriate domain via MDG.

When messages come in from the outside, they are received by the Network Interface (NI) and stored in the message buffer pool, which is stored on a page-by-page basis to index and manage messages. It generates a unique hash value

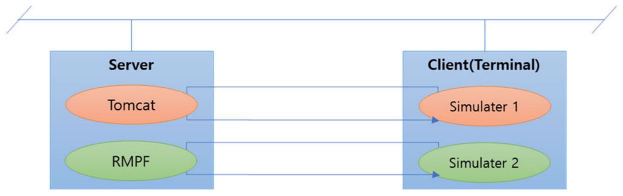

FIGURE 10. Test environment diagram.

TABLE 1. Test conditions.

Test case	1	2	3	4	Number of message
Test requirements	20 × 1000	40 × 500	60 × 333	80 × 250	20000

for each message. It passes this hash value address to the message converter (MC). The message converter goes to the hash value address and converts the stored raw message into an internal standard message (internal message form). After the conversion is complete, it sends the hash value address back to the Message Router (MR). The MR sends the hash value address to the corresponding SAU, which then goes to the hash value address to receive and process the converted internal message. It sends the completed internal message to its destination. For external service messages, they are sent through the MR, then the MC, then the NI to the external service. Since messages have a unique hash value, even if the message is processed and the value changes, the hash value remains unchanged and follows the message, ensuring the delivery of the message. In addition, since these processes are monitored in real time by the monitoring module, even if there is a failure, rapid recovery is performed according to the failure handling procedure, and the delivery of messages on the platform is guaranteed through this process. In addition, after the service system receives the authorized message through NI, the corresponding SAU of the MSP checks the authorization process of the message. If successful, the message is sent to the sending terminal to complete the message processing. DMPL also monitors the status of all components and application processes in each module through monitors. If a problem occurs due to an abnormal message, the recovery component handles the failure in real time and normalizes the system. In addition, because the units in each module are individually configured, if a failure occurs in one unit, it only affects that unit and does not affect the entire process because only that unit needs to be handled in the event of a failure. Fig. 9 is a flowchart of the message processing in the framework.

V. PERFORMANCE EVALUATION

We evaluate the proposed RMPF to prove that it performs well. We also evaluate the system by comparing RMPF and Apache Tomcat for middleware performance evaluation. Tomcat is a widely used general-purpose middleware that serves as a Java servlet container. For performance evaluation, two test environments are configured to compare and evaluate the performance under different test conditions.

A. EVALUATION ENVIRONMENT CONFIGURATION

Our evaluation environment consists of a server and a client (terminal) for testing. In addition, simulator is developed and utilized to replace the terminal to generate an accurate number of messages per unit time. The terminal is configured as a Linux environment to act as a card terminal. The simulator is developed, and used as a terminal model for card authorization system. The server consists of two software environments: the one is Tomcat as shown in Fig. 10, and the other is our RMPF for performance comparison. The client consists of two simulators each connected to one of software environments respectively.

The hardware specifications of the server system for testing are CPU: Intel(R) Xeon(R) CPU E5-2660 0 @ 2.20 GHz 8 core, RAM: 65776920 kB (65 GB), HDD: sda 1TB (for OS) and sdb 2TB (for monitor log collection). The software specifications are OS: CentOS 7.9, WAS: Apache Tomcat 9.0.64, Java 1.8.1_362, SQL-Lite 3.7.17, Logger, MRU, HIU, TIU and Log-Viewer.

In addition, the hardware specifications of the client (terminal) system are CPU: Intel(R) Xeon(R) CPU E5-2660 0 @ 2.20 GHz 8 core, RAM: 3819380 kB (3.8 GB), HDD: sda 40 GB (for OS) and sdb 5 GB (for monitor log collection). The software specifications are OS: Red Hat Enterprise Linux (RHEL) 8.6, WAS: Apache Tomcat 9.0.65 and Java 1.8.322.

B. EVALUATION METHODS

We compare our proposed RMPF with Tomcat system for 4 cases as in Table 1, where the number of terminals and messages varies, and measure the performance in terms of response time, execution time, CPU utilization and memory usage as shown in Table 2. The test sequence is to generate a message from the client, and send it to each platform in the server system. Each platform then processes the received messages, and sends the results back to the client. For each time period, we send messages simultaneously, and measure the processing time for a total of 20000 messages, that is, the maximum number of messages which can be processed without busy condition in Tomcat server. The reason why the number of messages processed cannot be increased further in this test is that while CPU utilization of RMPF has sufficient margin, that of Tomcat platform become close to the limit very fast.

The following metrics were used for testing. The test case was tested four times, and the test environment consisted of two platforms, Tomcat and RMPF, on one hardware to create the same environment. Number of message per second (sec) is the number of messages per second sent from the terminal to the platform. Total cases is the total number of messages sent per test case. Response time per message (sec) is the time for one message sent from the device to the platform,

TABLE 2. Summary of evaluation results.

Test case	Test environment	Number of message per second (sec)		Total message	Response time per message (sec)		Total execution time (sec)		CPU (%)	MEMORY Free / Total (KB)
		Min	Max		Min	Max	Min	Max		
1	Tomcat	57.4	57.8	20000	0.02	0.17	346.07	348.12	82.92	62192.4 / 64235.3 (use: 2042.9)
	RMPF	65.1	65.1	20000	0.03	2.16	307.05	307.09	0.70	39747.8 / 64235.3 (use: 24487.5)
2	Tomcat	61.7	62.6	20000	0.02	0.16	319.13	324.14	95.94	62155.8 / 64235.3 (use: 2079.5)
	RMPF	131.5	131.5	20000	0.03	0.16	152.07	152.09	1.11	39522.9 / 64235.3 (use: 24712.4)
3	Tomcat	60.7	61.2	19980	0	3.19	326.03	329.12	98.48	62155.8 / 64235.3 (use: 2103.0)
	RMPF	123.4	162.6	19980	0.02	2.16	123.08	162.05	0.96	39271.1 / 64235.3 (use: 2131.8)
4	Tomcat	60.4	61.7	20000	0	3.18	324.16	331.14	99.27	62103.5 / 64235.3 (use: 2131.8)
	RMPF	132.4	134.2	20000	0.03	2.14	149.04	151.15	0.92	38978.4 / 64235.3 (use: 25256.9)

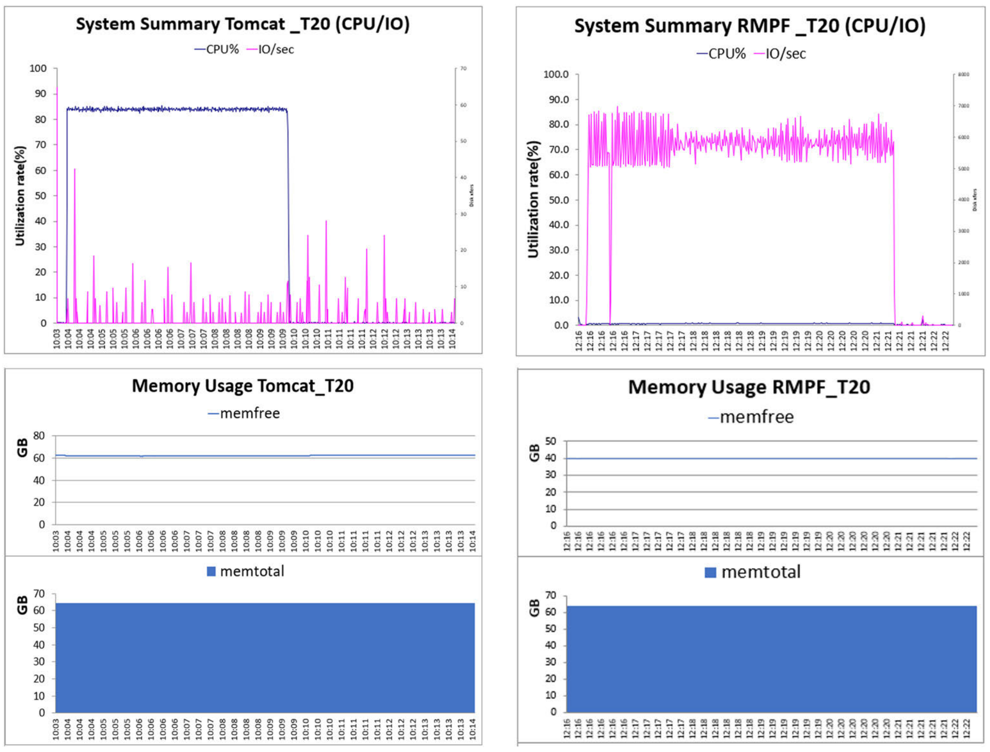

FIGURE 11. Case 1: (a) Tomcat CPU, MEMORY usage, (b) RMPF CPU, MEMORY usage.

processed by the platform, and received back from the device. Total execution time is the total time for all 20000 messages sent in one test case to be processed. CPU (%) is the percentage of CPU utilization in processing messages for each test case. Memory Free/Total (KB) is the amount of memory used and free memory for processing messages for each test case.

In test case 1, each of 20 terminals simultaneously sends 1000 messages, and in case 2, each of 40 terminals simultaneously sends 500 messages. Similarly, in case 3, each of 60 terminals simultaneously sends 333 messages simultaneously, and in case 4 each of 80 terminals simultaneously sends 250 messages. We also measure, and compare the CPU and memory usage of the server running the message processing in each environment.

Fig. 11 through Fig. 14, the vertical axis represents CPU utilization (%) and memory (GB), and the horizontal axis represents measurement time (seconds). The redline indicate I/O usage for reference. In Fig. 11, (a) shows the CPU utilization

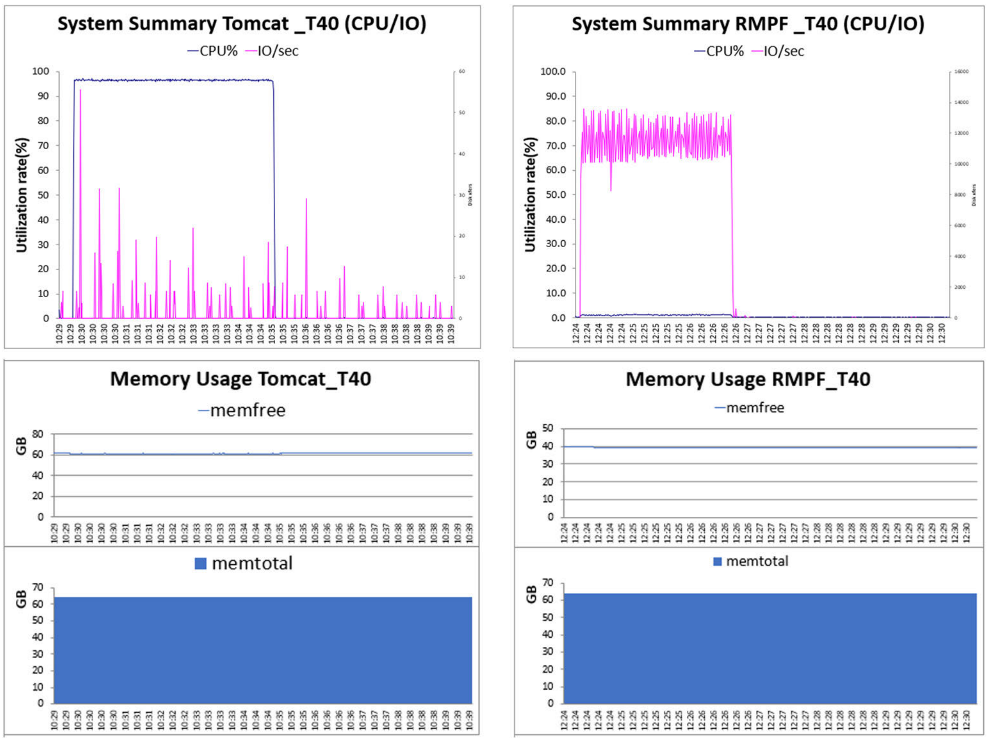

FIGURE 12. Case 2: Tomcat (a) CPU, MEMORY usage, RMPF (b) CPU, MEMORY usage.

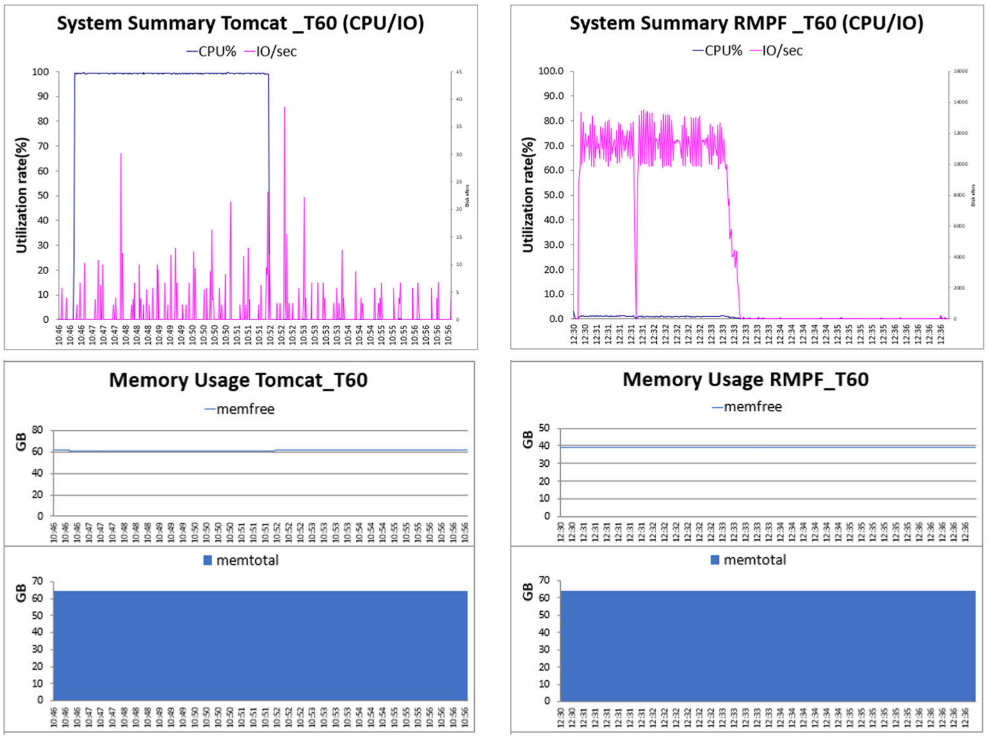

FIGURE 13. Case 3: Tomcat (a) CPU, MEMORY usage, RMPF (b) CPU, MEMORY usage.

and memory usage of Tomcat, and (b) those of RMPF respectively for case 1. In (a), the average CPU utilization is 82.92 % and the average memory usage is 2.0 GB on Tomcat for system software operations, platform and processor creation and message processing. In (b), the average CPU utilization is 0.7 %, and the average memory usage is 24.4 GB on RMPF

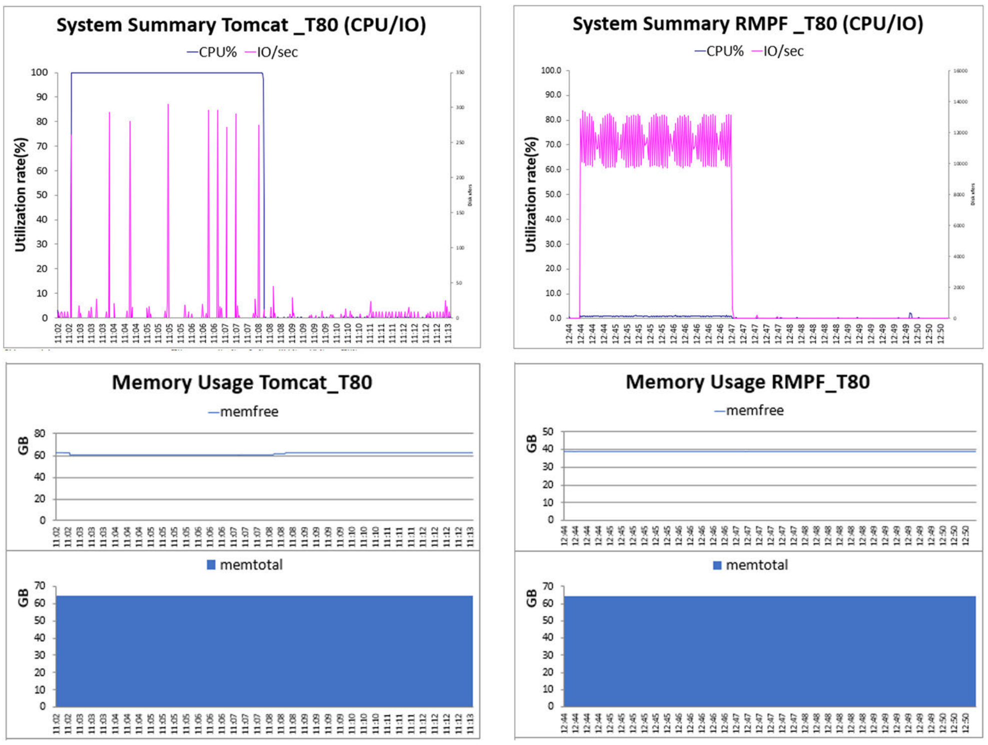

FIGURE 14. Case 4: Tomcat (a) CPU, MEMORY usage, RMPF (b) CPU, MEMORY usage.

for system software operations, platform and processor creation and message processing.

Fig. 12 (a) shows the test results of Tomcat for case 2, where CPU utilization is 95.94 % and the memory usage is 2.0 GB. Fig. 12 (b) shows the test results of RMPF for case 2, where CPU utilization is 1.11 % and the memory usage is 24.7 GB.

Fig. 13 (a) shows the test results of Tomcat for case 3, where CPU utilization is 98.48 %, and the memory usage is 2.1 GB, while Fig. 13 (b) shows those for RMPF, where CPU utilization is 0.96 %, and the memory usage is 24.9 GB.

Fig.14 (a) shows the test results of Tomcat for case 4, where CPU utilization is 99.27 %, and the memory usage is 2.1 GB, while (b) for those of RMPF, where CPU utilization is 0.92 %, and memory usage is 25.2 GB.

To summarize the evaluation results, for case 1, CPU utilization in RMPF is 118 times less than, and 22.4 GB more memory than the baseline Tomcat. For case 2, CPU utilization in RMPF is 86 times less than, and 22.6 GB more memory than the baseline Tomcat. For case 3, CPU utilization in RMPF is 103 times less than, and 22.8 GB more memory than the baseline Tomcat. For case 4, CPU utilization in RMPF is 106 times less than, and 23.1 GB more memory than the baseline Tomcat.

Therefore, the CPU utilization of RMPF is very lower than Tomcat, which shows its superiority over Tomcat. The reason for this is due to the fact that Tomcat initializes the environment, and creates new application process whenever the incoming message is received, while once our system sets up the environment, and runs application process as a daemon initially, it can continuously process messages on the running processes without any further setup time. Therefore, in RMPF, the CPU utilization is low, since the initially created process resides as a daemon, and hence SAU only just needs to invocate application process on CPU for the incoming message. On the other hand, Tomcat mainly uses process thread creation whenever a message comes in, a new process is created, and requires a lot of CPU operations to and from memory. Moreover, application process in SAU of our system can directly access, and process data stored in the buffer of message buffer pool dedicated to the process, while several processes in Tomcat should access the same common system queue.

Initially, the memory usage of RMPF is higher than that of Tomcat, since RMPF requires a message buffer pool during platform configuration, initial processor creation for monitoring, platform management, SAU execution, etc. However, after initialization, the memory usage for RMPF is small, since the created process only needs to process incoming messages. Therefore, even if the message throughput increases rapidly, the memory usage of RMPF does not increase rapidly, which enables efficient and stable system operation. The above performance evaluation results demonstrate the superiority of our proposed platform.

VI. CONCLUSION

In this paper, we have proposed a new general-purpose platform which provides real-time distributed message processing for multi-service domains. It supports multilayer architecture which consists of three layers: MDML, DMPL and MSPL, providing extensibility and flexibility by modularization of each layer which can be designed and updated independently. It innovatively combines the affordability of a general-purpose platform with the reliability and operational efficiency of a dedicated platform. RMPF provides the capability for fast message processing while achieving reliability by using the dedicated buffer in MBP created exclusively for each application process residing as a daemon for the incoming message, and then tracking the status of all messages in the buffer through monitor. It minimizes the waste of hardware resources due to memory fragmentation and frequent process creation arising in a general-purpose platform by keeping the application process in SAU with its own memory space in MBP, and providing efficient MBP management scheme for the dedicated buffer space. We also provide SAU which can be customized for different service type, and simplify the handling of multiple services by grouping the same tasks for each service.

Besides, it provides domain-based multi-services by supporting the grouping of RMPFs according to domains, and remove the system bottleneck by enabling the distribution of tasks to the proper domain. Moreover, it is further enhanced by various multi-service distributed management schemes such as multiservice API, recovery, monitor and multiservice deployment, each of which is implemented in a modular fashion that ensures both scalability and adaptability, making it suitable for a wide range of environments from traditional legacy systems to modern cloud infrastructures.

Nevertheless, there are always issues that need to be addressed. Our platform has many advantages and outperforms general-purpose platforms. However, we have not yet reached the peak performance of dedicated platforms. To improve this further, we need additional fault-tolerant clustering capabilities and better independent storage monitoring and management. We also recognize the need for a more in-depth comparison of platform monitoring, recovery and management capabilities. This will be the subject of future research.

From a broader perspective, this work is an important step in the development of a framework for real-time message processing. Our vision for this framework is to evolve and encompass not only existing domains, but also new domains such as transportation, healthcare, Blockchain, Non-fungible Token (NFT), and metaverses. In essence, this work is an extension of our future research. The goal is to build a universal messaging platform for multi-distributed real-time processing systems.

REFERENCES

[1] Z. Liu, D. Zhou, F. Lu, J. Fang, and L. Zhang, "AutoShape: Real-time shape-aware monocular 3D object detection," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 15621–15630.

[2] S. Braun, H. Gamper, C. K. A. Reddy, and I. Tashev, "Towards efficient models for real-time deep noise suppression," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Jun. 2021, pp. 656–660.

[3] Y. Wu, H.-N. Dai, H. Wang, Z. Xiong, and S. Guo, "A survey of intelligent network slicing management for industrial IoT: Integrated approaches for smart transportation, smart energy, and smart factory," *IEEE Commun. Surveys Tuts.*, vol. 24, no. 2, pp. 1175–1211, 2nd Quart., 2022.

[4] Z. Li, Y. Sun, L. Zhang, and J. Tang, "CTNet: Context-based tandem network for semantic segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 12, pp. 9904–9917, Dec. 2022.

[5] T. Sharvari and K. S. Nag, "A study on modern messaging systems–Kafka, RabbitMQ and NATS streaming," 2019, *arXiv:1912.03715*.

[6] G. van Dongen and D. Van den Poel, "Evaluation of stream processing frameworks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 8, pp. 1845–1858, Aug. 2020.

[7] A. Amiri, S. Rezaie, C. N. Manchón, and E. de Carvalho, "Distributed receiver processing for extra-large MIMO arrays: A message passing approach," *IEEE Trans. Wireless Commun.*, vol. 21, no. 4, pp. 2654–2667, Apr. 2022.

[8] T. A. Khan, M. Ugur, K. Nathella, D. Sunwoo, H. Litz, D. A. Jimenez, and B. Kasikci, "Whisper: Profile-guided branch misprediction elimination for data center applications," in *Proc. 55th IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2022, pp. 19–34.

[9] K. Kim, D. Seo, Y.-B. Jeon, S.-S. Han, D.-S. Park, and C.-S. Jeong, "Real time message process framework for efficient multi business domain routing," in *Advances in Computer Science and Ubiquitous Computing*. Singapore: Springer, 2018, pp. 271–278.

[10] J. Lee, S. Jeong, S. Song, K. Kim, H. Choi, Y. Kim, and H. Kim, "Occamy: Memory-efficient GPU compiler for DNN inference," in *Proc. 60th ACM/IEEE Design Autom. Conf. (DAC)*, Jul. 2023, pp. 1–6.

[11] O. Bell, C. Gill, and X. Zhang, "Hardware acceleration with zero-copy memory management for heterogeneous computing," in *Proc. IEEE 29th Int. Conf. Embedded Real-Time Comput. Syst. Appl. (RTCSA)*, Aug. 2023, pp. 28–37.

[12] I. S. Olmedo, N. Capodieci, J. L. Martinez, A. Marongiu, and M. Bertogna, "Dissecting the CUDA scheduling hierarchy: A performance and predictability perspective," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2020, pp. 213–225.

[13] R. V. W. Putra, M. A. Hanif, and M. Shafique, "ROMANet: Fine-grained reuse-driven off-chip memory access management and data organization for deep neural network accelerators," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 4, pp. 702–715, Apr. 2021.

[14] B. Asgari, R. Hadidi, J. Cao, D. E. Shim, S.-K. Lim, and H. Kim, "FAFNIR: Accelerating sparse gathering by using efficient near-memory intelligent reduction," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Feb. 2021, pp. 908–920.

[15] Y. Cai, X. Chen, L. Tian, Y. Wang, and H. Yang, "Enabling secure NVM-based in-memory neural network computing by sparse fast gradient encryption," *IEEE Trans. Comput.*, vol. 69, no. 11, pp. 1596–1610, Nov. 2020, doi: 10.1109/TC.2020.3017870.

[16] Y. Alotaibi, "A new meta-heuristics data clustering algorithm based on Tabu search and adaptive search memory," *Symmetry*, vol. 14, no. 3, p. 623, Mar. 2022.

[17] J. Tang, C. Yang, C. Feng, J. Li, X. Gu, and X. Jiang, "Energy cooperation optimization in residential microgrid with virtual storage technology," *Math. Problems Eng.*, vol. 2021, pp. 1–11, Jan. 2021.

[18] B. Ko, G. Gu, and H.-G. Kim, "Learning with memory-based virtual classes for deep metric learning," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 11772–11781.

[19] V. Bogatyrev and A. Derkach, "Evaluation of a cyber-physical computing system with migration of virtual machines during continuous computing," *Computers*, vol. 9, no. 2, p. 42, May 2020.

[20] I. Ataie and W. Yu, "SVAGC: Garbage collection with a scalable virtual address swapping technique," in *Proc. IEEE Int. Conf. Cluster Comput. (CLUSTER)*, Sep. 2022, pp. 357–368.

[21] L. Cui, K. He, Y. Li, P. Li, J. Zhang, G. Wang, and X. Liu, "SwapKV: A hotness aware in-memory key-value store for hybrid memory systems," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 1, pp. 917–930, Jan. 2023.

[22] S. Xu, Q. Wang, X. Wang, S. Wang, and T. T. Ye, "Multiplication through a single look-up-table (LUT) in CNN inference computation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 6, pp. 1916–1928, Jun. 2022.

[23] S. Boukhtache, B. Blaysat, M. Grédiac, and F. Berry, "Alternatives to bicubic interpolation considering FPGA hardware resource consumption," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 2, pp. 247–258, Feb. 2021.

[24] B. T. Hoffman and D. Reichhardt, "Recovery mechanisms for cyclic (Huff-n-Puff) gas injection in unconventional reservoirs: A quantitative evaluation using numerical simulation," *Energies*, vol. 13, no. 18, p. 4944, Sep. 2020.

[25] Y. Otsubo, A. Otsuka, M. Mimura, T. Sakaki, and H. Ukegawa, "O-glassesX: Compiler provenance recovery with attention mechanism from a short code fragment," in *Proc. Workshop Binary Anal. Res.*, 2020, pp. 1–12.

[26] R. Rajaraman, P. K. Kapur, and D. Kumar, "Determining software inter-dependency patterns for integration testing by applying machine learning on logs and telemetry data," in *Proc. 8th Int. Conf. Rel., INFOCOM Technol. Optim. (Trends Future Directions) (ICRITO)*, Jun. 2020, pp. 1080–1084.

[27] R. Chiramdasu, G. Srivastava, S. Bhattacharya, P. K. Reddy, and T. R. Gadekallu, "Malicious URL detection using logistic regression," in *Proc. IEEE Int. Conf. Omni-Layer Intell. Syst. (COINS)*, Aug. 2021, pp. 1–6.

[28] H. Pan, J. Liang, S. C. Liew, V. C. M. Leung, and J. Li, "Timely information update with nonorthogonal multiple access," *IEEE Trans. Ind. Informat.*, vol. 17, no. 6, pp. 4096–4106, Jun. 2021.

[29] A. Alahmadi and T. S. Chung, "RSLSP: An effective recovery scheme for flash memory leveraging shadow paging," *Electronics*, vol. 11, no. 24, p. 4126, Dec. 2022.

[30] K.-A. Tran, C. Sakalis, M. Själander, A. Ros, S. Kaxiras, and A. Jimborean, "Clearing the shadows: Recovering lost performance for invisible speculative execution through HW/SW co-design," in *Proc. ACM Int. Conf. Parallel Archit. Compilation Techn.*, Sep. 2020, pp. 31–42.

[31] Y. Seneviratne, K. Seemakhupt, S. Liu, and S. Khan, "NearPM: A near-data processing system for storage-class applications," 2022, *arXiv:2210.10094*.

[32] H. Du, Q. Li, R. Pan, T.-W. Kuo, and C. J. Xue, "Multi-granularity shadow paging with NVM write optimization for crash-consistent memory-mapped I/O," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Feb. 2023, pp. 108–121.

[33] A. Ioana, A. Korodi, and I. Silea, "Automotive IoT Ethernet-based communication technologies applied in a V2X context via a multi-protocol gateway," *Sensors*, vol. 22, no. 17, p. 6382, Aug. 2022.

[34] A. Ioana and A. Korodi, "DDS and OPC UA protocol coexistence solution in real-time and Industry 4.0 context using non-ideal infrastructure," *Sensors*, vol. 21, no. 22, p. 7760, Nov. 2021.

[35] H. Arnarson, B. Solvang, and B. Shu, "The application of open access middleware for cooperation among heterogeneous manufacturing systems," in *IEEE MTT-S Int. Microw. Symp. Dig.*, Jun. 2020, pp. 1–6.

[36] T. Wu, B. Wu, S. Wang, L. Liu, S. Liu, Y. Bao, and W. Shi, "Oops! It's too late. Your autonomous driving system needs a faster middleware," *IEEE Robot. Autom. Lett.*, vol. 6, no. 4, pp. 7301–7308, Oct. 2021.

[37] M. Jergler, K. Zhang, and H.-A. Jacobsen, "Multi-client transactions in distributed publish/subscribe systems," in *Proc. IEEE 38th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2018, pp. 120–131.

[38] V. Ashiwal, M. Majumder, and A. Zoitl, "Evaluation of middleware technologies for the PLC-service bus in IEC 61499," in *Proc. IEEE 27th Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2022, pp. 1–4.

[39] Y. Shi, J. Wong, H.-A. Jacobsen, Y. Zhang, and J. Chen, "Topic-oriented bucket-based fast multicast routing in SDN-like publish/subscribe middleware," *IEEE Access*, vol. 8, pp. 89741–89756, 2020.

[40] S. Sadeghianasl, A. H. M. T. Hofstede, S. Suriadi, and S. Turkay, "Collaborative and interactive detection and repair of activity labels in process event logs," in *Proc. 2nd Int. Conf. Process Mining (ICPM)*, Oct. 2020, pp. 41–48.

[41] D. R. K. M, J. Lee, E. Ko, S.-Y. Shin, J.-I. Namgung, S.-H. Yum, and S.-H. Park, "Underwater network management system in Internet of Underwater Things: Open challenges, benefits, and feasible solution," *Electronics*, vol. 9, no. 7, p. 1142, Jul. 2020.

[42] G. Chodak, G. Suchacka, and Y. Chawla, "HTTP-level e-commerce data based on server access logs for an online store," *Comput. Netw.*, vol. 183, Dec. 2020, Art. no. 107589.

[43] K. Beyer, R. Uola, K. Luoma, and W. T. Strunz, "Joint measurability in nonequilibrium quantum thermodynamics," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 106, no. 2, Aug. 2022, Art. no. L022101.

KYO-EUN KIM received the B.S. degree from the Department of Computer and Telecommunications, Seoul Cyber University, Seoul, South Korea, in 2015. He is currently pursuing the integrated M.S. and Ph.D. degrees in visual information processing with Korea University, Seoul. His research interests include artificial intelligence, middleware platforms, electronic payment systems, and distributed parallel computing.

SEONG-SOO HAN received the B.S. degree from Gyeongsang National University, South Korea, in 2000, the M.S. degree from Soonchunhyang University, South Korea, in 2005, and the Ph.D. degree from Korea University, South Korea, in 2019. He was a Professor with Soonchunhyang University, from 2018 to 2019. He is currently a Professor with the Division of Liberal Studies, Kangwon National University. His research interests include computer education, artificial intelligence, blockchain, and distributed parallel processing.

CHANG-SUNG JEONG (Member, IEEE) received the B.S. degree from Seoul National University, in 1981, and the M.S. and Ph.D. degrees from Northwestern University, in 1985 and 1987, respectively. He was an Assistant Professor with POSTECH, from 1987 to 1992. He was an Associate Researcher with UCSC, from 1998 to 2000. He is currently a Professor with the School of Electrical Engineering, Korea University. His research interests include distributed parallel computing, grid computing, ubiquitous computing, networked virtual computing, and development of highly intensive applications, such as stereo image processing and 3-D visualization on collaborative grid and ubiquitous computing environment.