International Journal of Innovative Computer Science and IT Research

Volume 02, Issue 03 – May-Jun 2026 • E-ISSN: 3067-1108 | P-ISSN: 3067-1094

Distributed Computing Models for High-Performance Data Processing

Dr. Laura Nieminen

Associate Professor, University of Turku, Finland

Abstract

The rapid growth of data generated by digital services, Internet of Things devices, scientific applications, enterprise systems, and intelligent technologies has created increasing demand for high-performance data processing architectures. Conventional centralized computing environments may encounter limitations when processing large datasets because of restricted computational capacity, memory constraints, storage bottlenecks, and communication overhead. Distributed computing provides an alternative by dividing computational and data-processing tasks across multiple interconnected processing nodes. This paper examines distributed computing models designed to support high-performance data processing and develops a conceptual framework for understanding how distributed architectures can improve scalability, parallelism, fault tolerance, and resource utilization. The study analyzes cluster computing, grid computing, cloud-based distributed processing, peer-to-peer architectures, edge-assisted processing, and hybrid distributed models. Particular attention is given to data partitioning, task scheduling, parallel execution, communication management, load balancing, fault recovery, and resource coordination. The proposed framework connects workload characteristics with appropriate distributed execution models and considers computational intensity, data volume, latency requirements, resource heterogeneity, and reliability requirements.

The analysis demonstrates that high-performance processing depends not only on the number of computing nodes but also on effective workload decomposition, communication efficiency, data locality, and scheduling strategies. Distributed data-processing frameworks can increase computational throughput by executing independent tasks concurrently, while data-local processing can reduce unnecessary movement across network infrastructure. However, increasing the number of processing nodes may also introduce synchronization and communication overhead, creating scalability limitations in highly interconnected workloads. The study further examines intelligent scheduling and adaptive resource allocation as mechanisms for responding to dynamic processing requirements. Challenges involving heterogeneous infrastructure, network congestion, fault management, consistency, security, and energy consumption are also discussed. The paper concludes that future high-performance distributed computing environments should combine scalable architectures with intelligent workload management, data-aware scheduling, adaptive resource allocation, and efficient communication mechanisms.

Keywords: Distributed computing, high-performance computing, data processing, parallel computing, cluster computing, cloud computing, distributed data processing, task scheduling, data partitioning, scalable computing

International Journal of Innovative Computer Science and IT Research

Volume 02, Issue 03 – May-Jun 2026 • E-ISSN: 3067-1108 | P-ISSN: 3067-1094

1. Introduction

The rapid expansion of digital information has significantly increased the volume, velocity, and complexity of data requiring computational processing. Modern applications generate data through enterprise platforms, scientific experiments, social systems, IoT devices, financial services, healthcare technologies, and intelligent applications. Processing these datasets using a single computing system can create substantial limitations because computational resources, memory, storage, and input/output capacity are inherently bounded. Distributed computing addresses these limitations by coordinating multiple processing nodes and dividing computational tasks across interconnected resources.

Distributed computing models allow large computational workloads to be decomposed into smaller tasks that can execute concurrently or independently. This parallel execution can significantly improve processing throughput when applications contain sufficient task-level or data-level parallelism. Cluster computing, cloud computing, grid computing, and distributed data-processing platforms provide different mechanisms for coordinating these resources. Each model offers different advantages in terms of scalability, resource availability, communication structure, fault tolerance, and management complexity.

High-performance data processing depends heavily on effective data partitioning and workload distribution. Large datasets can be divided into partitions and processed simultaneously across multiple nodes. However, inefficient partitioning may create load imbalance in which some nodes complete their work earlier while others remain overloaded. Communication among processing nodes can also become a major bottleneck when tasks require frequent data exchange. Therefore, distributed systems must balance computational parallelism with communication efficiency and data locality.

Cloud computing has further expanded distributed data-processing capabilities by providing dynamically scalable infrastructure. Cloud platforms can allocate additional computing resources when processing demand increases and release them when demand declines. Distributed data-processing frameworks can consequently support large-scale analytical workloads without requiring organizations to maintain permanent high-capacity infrastructure. However, cloud-based processing may introduce network latency, resource contention, data-transfer costs, and dependency on external infrastructure.

The objective of this paper is to examine distributed computing models for high-performance data processing and develop a conceptual framework for selecting and optimizing distributed execution strategies. The study focuses on workload decomposition, data partitioning, task scheduling, resource allocation, data locality, communication management, and fault tolerance. It also examines intelligent and adaptive mechanisms that can improve distributed processing under changing workloads. The paper identifies important challenges and future research directions for building scalable, efficient, reliable, and intelligent distributed data-processing environments.

2. Objectives

2.1 Primary Objective

The primary objective of this study is to examine distributed computing models that support high-performance data processing and develop a conceptual framework for improving computational scalability, parallel execution, resource utilization, and processing efficiency across interconnected computing environments.

2.2 Development of a Distributed Processing Framework

The study aims to establish a structured framework that connects workload characteristics, data partitioning, task scheduling, processing resources, and communication requirements. The framework is intended to support efficient distribution of computational tasks across heterogeneous processing nodes.

2.3 Optimization of Parallel Data Processing

Another objective is to analyze mechanisms for improving parallel execution through workload decomposition, data locality, load balancing, distributed scheduling, and adaptive resource allocation. These mechanisms can reduce processing bottlenecks and improve overall system throughput.

2.4 Identification of Distributed Processing Challenges

The study also identifies challenges involving communication overhead, synchronization, heterogeneous infrastructure, fault tolerance, data consistency, security, energy consumption, and scalability. Understanding these factors is essential for designing reliable distributed computing models for large-scale data processing.

3. Review of Literature

3.1 Foundations of Distributed Computing

Distributed computing involves multiple autonomous computing resources that coordinate to perform computational tasks. Tanenbaum and Van Steen described distributed systems as collections of independent computers that appear to users as a coherent system. The distributed model enables computational resources to be combined for larger workloads while providing opportunities for fault tolerance and scalability.

Cluster and grid computing established important foundations for high-performance distributed processing. Cluster architectures typically connect computing nodes within controlled infrastructure, while grid computing enables geographically distributed resources to participate in computational tasks. These approaches demonstrate that workload decomposition and coordinated resource management are fundamental to distributed performance.

3.2 Distributed Data Processing Frameworks

Large-scale data processing has increasingly relied on distributed programming models such as MapReduce. Dean and Ghemawat introduced MapReduce as a programming model capable of processing large datasets across clusters by dividing computation into map and reduce stages. Hadoop subsequently provided an open-source ecosystem for distributed storage and computation.

Spark and related in-memory processing frameworks further expanded distributed data analytics by reducing repeated disk access for workloads involving iterative computation. These frameworks demonstrate the importance of data locality, parallel execution, task scheduling, and efficient memory management in high-performance data processing.

3.3 Cloud, Edge, and Intelligent Distributed Computing

Cloud computing provides elastic infrastructure for distributed data processing, allowing computational resources to scale according to workload requirements. Edge computing extends distributed processing

toward data-generation locations, potentially reducing latency and unnecessary data movement. These developments have created increasingly heterogeneous distributed environments.

Intelligent resource management can improve these environments by predicting workload requirements, optimizing task placement, and dynamically adjusting resource allocation. Machine learning and adaptive scheduling can help identify suitable execution locations based on computational requirements, data locality, network conditions, and available resources. However, intelligent mechanisms must account for their own computational and communication overhead.

4. Methodology

4.1 Research Design

This study adopts a conceptual and analytical research design for examining distributed computing models used in high-performance data processing. The methodology synthesizes principles from distributed systems, parallel computing, cloud computing, data-intensive processing, workload scheduling, and resource management. Rather than presenting fabricated experimental measurements, the study develops a structured framework for analyzing distributed processing mechanisms and their relationships with performance objectives.

The analytical model considers data volume, computational intensity, task dependency, communication requirements, processing capacity, network conditions, and fault tolerance as major variables. These dimensions are used to examine how distributed computing models can support parallel execution while minimizing communication and synchronization overhead. The framework also considers scalability and resource efficiency as central performance objectives.

4.2 Distributed Data Processing Architecture

The proposed architecture consists of five major layers: data ingestion, data partitioning, task scheduling, distributed execution, and result aggregation. The data-ingestion layer receives datasets from applications and connected systems. Data partitioning divides large datasets into manageable units, while the scheduling layer assigns processing tasks to suitable nodes according to resource availability and workload characteristics.

The distributed execution layer performs computation across multiple processing nodes. Data locality is considered when possible to reduce unnecessary movement between storage and computation resources. The result-aggregation layer combines outputs from distributed tasks into application-level results. Monitoring mechanisms operate across the architecture to identify overloaded nodes, failed tasks, and changing resource conditions.

4.3 Key Methodological Components

The methodology incorporates the following components:

- **Data partitioning:** Dividing large datasets into manageable processing units.
- **Task decomposition:** Breaking complex computations into executable tasks.
- **Distributed scheduling:** Assigning tasks to appropriate processing nodes.
- **Parallel execution:** Executing independent tasks concurrently.
- **Data locality:** Processing data near its storage location where practical.
- **Load balancing:** Distributing workloads across available resources.

- **Fault management:** Detecting failed tasks and initiating recovery.
- **Resource scaling:** Adjusting computing capacity according to workload requirements.
- **Result aggregation:** Combining distributed outputs into coherent results.
- **Performance monitoring:** Tracking processing and communication conditions.

These components form an iterative processing cycle. Workloads are first analyzed and divided into executable units, after which tasks are scheduled across available resources. Execution is monitored continuously, allowing the system to respond to node failures, workload imbalance, or resource changes. The resulting outputs are then aggregated and delivered to the target application.

5. Analysis

5.1 Distributed Computing Models

Different distributed computing models are suitable for different workload characteristics. Cluster systems are particularly appropriate for tightly controlled environments with high-speed interconnections, while cloud-based models provide greater elasticity.

Grid systems can utilize geographically distributed resources, whereas edge-based models are useful when processing must occur near data sources.

Table 1: Distributed Computing Models for High-Performance Data Processing

Computing Model	Resource Structure	Major Strength	Suitable Processing Scenario
Cluster Computing	Closely connected dedicated nodes	High communication efficiency	Scientific and analytical workloads
Grid Computing	Geographically distributed resources	Resource sharing	Large computational projects
Cloud Computing	Elastic virtualized infrastructure	Scalability and flexibility	Variable large-scale workloads
Peer-to-Peer Computing	Decentralized participating nodes	Distributed resource utilization	Decentralized applications
Edge Computing	Processing near data sources	Low latency	Real-time data processing
Hybrid Distributed Computing	Combination of multiple environments	Flexible resource coordination	Complex heterogeneous workloads

Interpretation: Table 1 shows that distributed computing models provide different approaches to high-performance data processing. Cluster computing emphasizes tightly integrated resources and efficient communication, whereas cloud computing provides elastic capacity for variable workloads.

Grid computing can coordinate geographically distributed resources, while edge computing minimizes communication distance between data sources and processing resources. Hybrid models

provide greater flexibility by combining multiple environments. Therefore, model selection should depend on workload characteristics, latency requirements, data distribution, resource availability, and communication patterns.

5.2 Processing and Optimization Mechanisms

High-performance distributed processing depends on mechanisms that effectively divide workloads and coordinate execution. Data partitioning determines how information is distributed, while task scheduling determines where computations are executed.

Load balancing prevents individual nodes from becoming bottlenecks, and data locality reduces unnecessary network transfers.

Table 2: Key Mechanisms for High-Performance Distributed Data Processing

Mechanism	Primary Function	Performance Contribution	Main Consideration
Data Partitioning	Divides large datasets	Enables parallel processing	Partition balance
Task Decomposition	Breaks workloads into tasks	Increases execution parallelism	Task dependency
Distributed Scheduling	Assigns tasks to nodes	Improves resource utilization	Scheduling overhead
Data Locality	Places computation near data	Reduces data transfer	Placement flexibility
Load Balancing	Distributes computational work	Reduces processing bottlenecks	Monitoring requirements
In-Memory Processing	Keeps active data in memory	Reduces repeated storage access	Memory capacity
Fault Recovery	Restarts failed tasks	Improves system reliability	Recovery overhead
Dynamic Scaling	Adjusts processing capacity	Supports changing workloads	Provisioning delay
Result Aggregation	Combines distributed outputs	Produces final results	Synchronization cost

Interpretation: The mechanisms in Table 2 operate at different stages of the distributed processing lifecycle. Partitioning and task decomposition create opportunities for parallel execution, while scheduling determines how effectively available resources are used. Data locality can reduce communication overhead, and load balancing prevents uneven task distribution.

In-memory processing can accelerate iterative workloads when sufficient memory is available. Fault recovery maintains reliability, while dynamic scaling enables systems to respond to changing demand. Effective high-performance processing therefore requires coordinated use of multiple mechanisms rather than reliance on a single optimization technique.

5.3 Factors Affecting Distributed Processing Performance

Communication overhead is one of the most significant factors influencing distributed processing performance. When computational tasks require frequent information exchange, the network may become a bottleneck even when sufficient processing resources are available. High communication requirements can also increase synchronization delays. Data-aware scheduling and locality-oriented execution can reduce these limitations by minimizing unnecessary movement.

Load imbalance represents another major performance issue. If one processing node receives substantially more computational work than others, overall execution time may be determined by the slowest node. Dynamic scheduling and workload redistribution can address this problem. However, redistributing tasks can introduce additional communication and coordination costs.

Scalability also depends on workload characteristics. Applications with independent tasks can scale efficiently because additional nodes can execute tasks concurrently. Applications with strong task dependencies may achieve lower scalability because synchronization limits the degree of parallel execution. Consequently, distributed processing architectures should consider the parallel structure of workloads before allocating additional resources.

6. Discussion

6.1 Parallelism and Computational Scalability

Distributed computing improves high-performance processing primarily by enabling computational parallelism. Large workloads can be divided into smaller tasks and executed concurrently across multiple resources. When tasks are sufficiently independent, increasing the number of processing nodes can reduce overall execution time and improve throughput.

However, scalability is not unlimited. Communication, synchronization, task coordination, and data movement may increase as additional nodes are introduced. Consequently, high-performance distributed systems must identify an appropriate balance between computational parallelism and coordination overhead. Effective scheduling and task decomposition are therefore central to scalable distributed processing.

6.2 Data Locality and Intelligent Scheduling

Data locality is particularly important in data-intensive applications because moving large datasets across networks can consume significant bandwidth and processing resources. Scheduling computation near the required data can reduce transfer overhead and improve processing efficiency. Distributed frameworks can therefore benefit from data-aware task placement.

Intelligent scheduling can further improve resource utilization by analyzing workload requirements and infrastructure conditions. Predictive mechanisms can estimate computational demand and identify potential bottlenecks before execution. Adaptive schedulers can then redistribute tasks or

allocate additional resources. Lightweight machine-learning models may provide useful predictions without introducing excessive management overhead.

6.3 Reliability and Resource Coordination

High-performance distributed systems must remain functional despite individual node or task failures. Because distributed architectures contain multiple processing components, the probability of component failure can increase with system scale. Replication, checkpointing, task recomputation, and automated recovery can help maintain reliable processing.

Resource coordination also becomes increasingly important in heterogeneous environments. Processing nodes may differ in computational capacity, memory, storage, and network connectivity. Treating all resources as identical can lead to inefficient scheduling. Performance-aware resource management should therefore consider node capabilities and workload requirements when assigning tasks.

7. Challenges

A major challenge in distributed data processing is managing communication and synchronization overhead. Parallel tasks often need to exchange intermediate information, coordinate execution stages, or synchronize their progress. As the number of processing nodes increases, these interactions can become more complex. Excessive communication may offset the performance benefits gained from parallel execution, particularly for workloads with strong dependencies.

Heterogeneous infrastructure presents another challenge. Modern distributed environments may combine physical servers, virtual machines, containers, cloud resources, and edge devices with significantly different capabilities. Resource allocation strategies designed for homogeneous clusters may not perform effectively in such environments. Scheduling systems must therefore recognize differences in computational capacity, memory, storage, network conditions, and energy characteristics.

Fault tolerance and data consistency also create significant challenges. Distributed systems must continue processing when individual nodes fail, while maintaining correct results and avoiding unnecessary duplication. Recovery mechanisms may introduce additional computation and communication overhead. Furthermore, sensitive datasets processed across multiple infrastructure domains require appropriate security controls, access management, and data-protection mechanisms.

8. Future Research

Future research should investigate intelligent distributed schedulers capable of combining workload prediction, resource monitoring, and data locality information. Machine-learning models could predict task execution time, resource demand, and communication requirements, enabling more accurate task placement. Reinforcement learning could also be used to develop adaptive scheduling policies that respond to changing infrastructure conditions. Research should emphasize computationally lightweight intelligence suitable for large distributed systems.

Another promising direction is unified computing across cloud, edge, and high-performance infrastructures. Future applications may dynamically distribute processing tasks among centralized cloud resources, local edge nodes, and dedicated high-performance computing systems. Intelligent orchestration could select execution locations according to latency, computational intensity, data

locality, energy availability, and cost. Such hybrid architectures could support real-time and large-scale analytical workloads simultaneously.

Future distributed systems should also investigate energy-aware high-performance processing. Increasing the number of active computing nodes can improve throughput but may significantly increase energy consumption. Energy-aware scheduling could consolidate workloads during periods of low demand and distribute them during computationally intensive periods. Research should further explore sustainable distributed architectures that balance processing performance, energy efficiency, reliability, and resource availability.

9. Conclusion

Distributed computing models provide an important foundation for high-performance data processing by enabling large workloads to be divided across multiple interconnected computing resources. The growing volume and complexity of digital data have made centralized processing increasingly difficult for computationally intensive applications. This paper examined cluster, grid, cloud, peer-to-peer, edge, and hybrid distributed computing models and developed a conceptual framework incorporating data ingestion, partitioning, task decomposition, scheduling, distributed execution, and result aggregation. The analysis demonstrated that high-performance processing depends not simply on increasing the number of computing nodes but on efficiently coordinating computational resources, data, communication, and workload dependencies.

The study further examined data partitioning, distributed scheduling, data locality, load balancing, in-memory processing, fault recovery, dynamic scaling, and result aggregation as important mechanisms for improving distributed processing performance. Data locality can reduce unnecessary data movement, while balanced task distribution can prevent individual nodes from becoming computational bottlenecks. Dynamic resource allocation can support variable workloads, and fault-recovery mechanisms can maintain system reliability. However, these benefits must be balanced against synchronization, communication, resource-management, and recovery overhead. Distributed processing is therefore most effective when the architecture and scheduling strategy are matched to the computational and communication characteristics of the workload.

Future distributed computing environments are likely to become increasingly heterogeneous as cloud, edge, high-performance computing, containerized infrastructure, and specialized processing resources converge. This evolution will require intelligent scheduling and orchestration mechanisms capable of making context-aware decisions across diverse computing environments. Predictive analytics, reinforcement learning, data-aware scheduling, adaptive resource allocation, and energy-aware processing represent promising research directions. The development of such mechanisms can enable distributed systems to dynamically match computational workloads with suitable resources while controlling communication and energy costs. By integrating scalability, performance, reliability, and intelligent resource coordination, distributed computing models can provide a robust technological foundation for next-generation high-performance data-processing applications.

References

1. Dean, J., & Ghemawat, S. (2008). MapReduce: Simplified data processing on large clusters. *Communications of the ACM, 51*(1), 107–113. https://doi.org/10.1145/1327452.1327492

2. Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S., & Stoica, I. (2012). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation*, 15–28.
3. Zaharia, M., Das, T., Li, H., Hunter, T., Shenker, S., & Stoica, I. (2013). Discretized streams: Fault-tolerant streaming computation at scale. *Proceedings of the 24th ACM Symposium on Operating Systems Principles*, 423–438.
4. Armbrust, M., Xin, R. S., Lian, C., Huai, Y., Liu, D., Bradley, J. K., Meng, X., Kaftan, T., Franklin, M. J., Ghodsi, A., & Zaharia, M. (2015). Spark SQL: Relational data processing in Spark. *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, 1383–1394. https://doi.org/10.1145/2723372.2742797
5. Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I. (2010). Spark: Cluster computing with working sets. *Proceedings of the 2nd USENIX Workshop on Hot Topics in Cloud Computing*.
6. White, T. (2015). *Hadoop: The definitive guide* (4th ed.). O'Reilly Media.
7. Vavilapalli, V. K., Murthy, A. C., Douglas, C., Agarwal, S., Konar, M., Evans, R., Graves, T., Lowe, J., Shah, H., Seth, S., Saha, B., Curino, C., O'Malley, O., Radia, S., Reed, B., & Baldeschwieler, E. (2013). Apache Hadoop YARN: Yet another resource negotiator. *Proceedings of the 4th Annual Symposium on Cloud Computing*, Article 5. https://doi.org/10.1145/2523616.2523633
8. Carbone, P., Katsifodimos, A., Ewen, S., Markl, V., Haridi, S., & Tzoumas, K. (2015). Apache Flink: Stream and batch processing in a single engine. *IEEE Data Engineering Bulletin*, *38*(4), 28–38.
9. Akidau, T., Bradshaw, R., Chambers, C., Chernyak, S., Fernández-Moctezuma, R. J., Lax, R., McVeety, S., Mills, D., Perry, F., Schmidt, E., & Whittle, S. (2015). The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. *Proceedings of the VLDB Endowment*, *8*(12), 1792–1803.
10. Sakr, S., Liu, A., & Fayoumi, A. G. (2013). The family of MapReduce and large-scale data processing systems. *ACM Computing Surveys*, *46*(1), Article 11. https://doi.org/10.1145/2522968.2522979
11. Ghemawat, S., Gobioff, H., & Leung, S.-T. (2003). The Google file system. *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, 29–43. https://doi.org/10.1145/945445.945450
12. Dean, J., Ghemawat, S., & others. (2010). MapReduce: A flexible data processing tool. *Communications of the ACM*, *53*(1), 72–77.
13. Low, Y., Gonzalez, J., Kyrola, A., Bickson, D., Guestrin, C., & Hellerstein, J. M. (2012). Distributed GraphLab: A framework for machine learning and data mining in the cloud. *Proceedings of the VLDB Endowment*, *5*(8), 716–727.
14. Malewicz, G., Austern, M. H., Bik, A. J. C., Dehnert, J. C., Horn, I., Leiser, N., & Czajkowski, G. (2010). Pregel: A system for large-scale graph processing. *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, 135–146. https://doi.org/10.1145/1807167.1807184

15. Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S., & Stoica, I. (2012). Fast and interactive analytics over Hadoop data with Spark. *USENIX Login*, *37*(4), 20–25.
16. Herodotou, H., Dong, F., & Babu, S. (2011). No one (cluster) size fits all: Automatic cluster sizing for data-intensive analytics. *Proceedings of the 2nd ACM Symposium on Cloud Computing*, Article 18. https://doi.org/10.1145/2038916.2038934
17. Zaharia, M., Konwinski, A., Joseph, A. D., Katz, R., & Stoica, I. (2008). Improving MapReduce performance in heterogeneous environments. *Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation*, 29–42.
18. Shvachko, K., Kuang, H., Radia, S., & Chansler, R. (2010). The Hadoop distributed file system. [1] 2010. IEEE 26th Symposium on Mass Storage Systems and Technologies, 1 [2] 10. https://doi.org/10.1109/MSST.2010.5496972
19. Czarnul, P., Antal, M., Baniata, H., Griebler, D., Kertesz, A., Kessler, C. W., Kouloumpris, A., Kovačić, S., Markus, A., Michael, M. K., Nikolaou, P., Öz, I., Prodan, R., & others. (2025). Optimization of resource-aware parallel and distributed computing: A review. *The Journal of Supercomputing*, *81*, Article 848. https://doi.org/10.1007/s11227-025-07295-7
20. Moka, A. K., & Lagisetty, S. K. (2025). Comparative analysis of big data processing frameworks: Python with MPI vs. Apache Spark. [3] 2025. IEEE SoutheastCon, 149– [4] 154. https://doi.org/10.1109/SOUTHEASTCON56624.2025.10971632