WCRG Global Research
WORLD COMET RESEARCH GROUP

International Journal of Artificial Intelligence & Digital Transformation
Vol. 1, No. 1, 2026
Doi: 10.XXXX/XXXXXXXX/IJAIDT-V1I1P103
PP. 22-28

Original Article

AI-Driven Dependency Analysis for Migrating Monolithic Applications to Microservices Architecture

M. Riyaz Mohammed

Assistant professor, Department of IT, Jamal Mohammed College (Autonomous), Tiruchirappalli, Tamil Nadu, India.

Received: 02-12-2025 Revised: 27-12-2025 Accepted: 30-12-2025 Published: 04-01-2026

ABSTRACT

Microservices Architecture (MSA) has become a de-facto standard for designing cloud-native enterprise applications due to its efficient infrastructure setup, service availability, elastic scalability, dependability, and enhanced security. Transitioning existing monolithic systems to microservices is essential to leverage these benefits. However, manual decomposition of large-scale applications is labor-intensive and prone to errors. AI-based systems offer promising solutions for automating this process. This paper introduces CARGO (Context-sensitive lAbel pRopaGatiOn), a novel un-/semi-supervised partition refinement technique that utilizes a context- and flow-sensitive system dependency graph of monolithic applications. CARGO refines and enhances the partitioning quality of existing microservice partitioning algorithms. Experiments demonstrate that CARGO improves partition quality, reduces distributed transactions, and enhances performance metrics such as latency and throughput in microservice applications.

KEYWORDS

Microservices Architecture, Monolithic Applications, AI-Driven Dependency Analysis, Application Migration, System Dependency Graph, Partition Refinement, Distributed Transactions, Performance Optimization.

1. INTRODUCTION

1.1. Overview of Microservices Architecture and its Advantages

Microservices Architecture (MSA) is a design approach that structures an application as a collection of loosely coupled, independently deployable services. Each service is focused on a specific business function and communicates with others through APIs. This modularity offers several advantages:

- Scalability: Individual services can be scaled independently based on demand, optimizing resource utilization and performance.
- Fault Isolation: Failures in one service are contained, preventing system-wide outages and enhancing overall reliability.
- Flexibility in Technology Stack: Teams can select the most suitable technologies for each service, fostering innovation and efficiency.
- Independent Deployment: Services can be deployed independently, facilitating continuous integration and delivery, and reducing time-to-market.

1.2. Challenges in Migrating Monolithic Applications to Microservices

Transitioning from a monolithic architecture to microservices presents several challenges:

- Complex Decomposition: Identifying appropriate boundaries for microservices within a monolithic codebase is intricate and requires deep understanding of the application's functionality.
- Data Management: Ensuring data consistency and managing transactions across distributed services can be complex, necessitating careful design and implementation.
- Operational Overhead: Managing multiple services increases operational complexity, including deployment, monitoring, and maintenance efforts.
- Inter-Service Communication: Establishing reliable and efficient communication between services is crucial, as network latency and failures can impact performance and reliability.

1.3. Role of AI in Automating the Migration Process

Artificial Intelligence (AI) plays a pivotal role in automating the migration from monolithic applications to microservices:

- Dependency Analysis: AI can analyze codebases to identify dependencies and interactions, aiding in the identification of natural boundaries for microservices.
- Refactoring Assistance: AI-driven tools can suggest code refactoring opportunities, streamline the decomposition process, and ensure that microservices are cohesive and loosely coupled.
- Performance Optimization: By analyzing system performance, AI can recommend optimizations and predict the impact of changes, ensuring that the migrated architecture meets performance requirements.
- Continuous Improvement: AI facilitates continuous learning from migration efforts, refining models and strategies for future migrations.

2. BACKGROUND AND RELATED WORK

2.1. Existing Methodologies for Application Decomposition

Several methodologies have been developed to assist in decomposing monolithic applications:

- Spatio-Temporal Decomposition: This approach leverages business use cases and runtime call relations to create functionally cohesive partitions. Tools like Mono2Micro have implemented this technique, demonstrating its effectiveness in decomposing Java applications.
- Language Model-Based Approaches: Recent studies have employed large language models to generate vector representations of monolithic components, which are then clustered to form

microservices. This method utilizes contrastive learning to enhance the quality of decomposition.

2.2. Limitations of Current AI-Based Approaches

Despite advancements, current AI-based decomposition approaches face limitations:

- Insufficient Representation: Some methods struggle to capture the full complexity of application dependencies, leading to suboptimal decomposition quality.
- External Resource Dependencies: Many approaches fail to adequately model dependencies on external resources like databases, which can result in architectures that require complex distributed transactions or exhibit monolithic characteristics.

2.3. Need for Modeling Dependencies on External Resources Like Databases

Incorporating knowledge of external resource dependencies is crucial for effective decomposition:

- Data Consistency: Understanding interactions with databases ensures that data consistency is maintained across services.
- Performance Optimization: Awareness of database dependencies allows for the design of services that minimize latency and optimize throughput.
- Transaction Management: Proper modeling helps in designing services that handle transactions efficiently, reducing the need for complex distributed transactions.

3. CARGO: CONTEXT-SENSITIVE LABEL PROPAGATION

3.1. Detailed Explanation of the CARGO Technique

CARGO (Context-sensitive lAbel pRopaGatiOn) is an AI-guided partition refinement technique designed to enhance the decomposition of monolithic applications into microservices:

- Context and Flow Sensitivity: CARGO analyzes the application's system dependency graph, considering both the context of operations and data flow to identify natural service boundaries.
- Unsupervised and Semi-Supervised Modes: CARGO operates in both unsupervised and semi-supervised modes, allowing it to adapt to varying levels of available training data and developer input.

3.2. Construction of the System Dependency Graph

The system dependency graph is central to CARGO's analysis:

- Node Representation: Each node represents a component or module within the application.
- Edge Representation: Edges denote dependencies and interactions between components, capturing both data flow and control flow.
- **Granularity**: The graph captures dependencies at a granular level, enabling precise identification of cohesive service boundaries.

3.3. Unsupervised and Semi-Supervised Modes of Operation

CARGO's flexibility allows it to function under different conditions:

- Unsupervised Mode: Without labeled training data, CARGO employs clustering techniques to identify patterns and suggest service boundaries based on the dependency graph.
- Semi-Supervised Mode: When some labeled data is available, CARGO refines its suggestions by incorporating this information, improving the accuracy of the decomposition.

By addressing the complexities of dependency modeling and offering flexible operation modes, CARGO significantly enhances the process of migrating monolithic applications to microservices architecture.

4. INTEGRATION WITH EXISTING PARTITIONING TECHNIQUES

4.1. Overview of State-of-the-Art Microservice Partitioning Algorithms

In the evolution from monolithic to microservice architectures, several partitioning algorithms have been developed to facilitate effective decomposition. Mono2Micro, for instance, employs spatio-temporal decomposition, leveraging business use cases and runtime call relations to create functionally cohesive partitions. This approach has demonstrated significant improvements over traditional methods. Similarly, the DEEPLY algorithm enhances partitioning by extending the CO-GCN deep learning model, incorporating novel loss functions and hyper-parameter optimization to achieve stable and superior partitioning across diverse datasets and objectives. Additionally, ModSoft-HP introduces fuzzy microservices placement within Kubernetes environments, allowing for flexible service distribution based on predefined thresholds, which aids in balancing load and optimizing resource utilization.

4.2. Methodology for Augmenting These Algorithms Using CARGO

CARGO (Context-sensitive lAbel pRopaGatiOn) enhances existing partitioning algorithms by refining partition boundaries through a context- and flow-sensitive analysis of the system's dependency graph. This refinement aims to improve cohesion within services and reduce inter-service coupling. By integrating CARGO, algorithms like Mono2Micro, DEEPLY, and ModSoft-HP can achieve more precise decompositions, minimizing issues such as distributed monoliths and complex distributed transactions. The iterative nature of CARGO's refinement process allows it to adapt to various partitioning strategies, making it a versatile tool for augmenting diverse decomposition approaches.

4.3. Case Studies and Applications on Java EE Applications

Practical applications of CARGO have been demonstrated through case studies on Java EE applications, including Daytrader and JPetStore. These studies involved applying CARGO to augment existing partitioning algorithms, resulting in significant improvements in partition quality. The enhancements led to a substantial reduction in distributed transactions and notable performance gains, with latency decreasing by 11% and throughput increasing by 120% on average. These case studies underscore the effectiveness of integrating CARGO into the migration process, highlighting its ability to optimize both the structural and operational aspects of microservice architectures.

5. EXPERIMENTAL EVALUATION

5.1. Metrics for Assessing Partition Quality

Evaluating the quality of microservice partitions involves several key metrics. Structural Modularity (SM) measures the cohesion within a partition and the coupling between partitions, with higher values indicating better modularity. The Interface Number (IFN) assesses the number of interfaces exposed by a service, with fewer interfaces generally being preferable. Additionally, metrics like transaction purity, latency, and throughput provide insights into the operational efficiency of the partitioned system. These metrics collectively offer a comprehensive assessment of partition quality, guiding the optimization of microservice architectures.

5.2. Impact of CARGO on Reducing Distributed Transactions

The integration of CARGO has a profound impact on minimizing distributed transactions within microservice architectures. By refining partition boundaries based on a detailed analysis of system dependencies, CARGO ensures that services with high interdependencies are colocated, thereby reducing the need for cross-service transactions. This reduction not only simplifies transaction management but also enhances system performance by decreasing latency and resource consumption associated with distributed transactions. Empirical studies have demonstrated that applying CARGO leads to a substantial decrease in distributed transactions, contributing to more efficient and reliable microservice implementations.

5.3. Performance Improvements in Terms of Latency and Throughput

The application of CARGO has been associated with significant performance enhancements in microservice architectures. By optimizing the decomposition process to align with system dependencies and interactions, CARGO reduces the overhead caused by inter-service communication and transaction management. This optimization leads to measurable improvements in latency, with reductions of up to 11%, and substantial increases in throughput, with gains averaging around 120%. These performance metrics highlight the efficacy of CARGO in transforming monolithic applications into responsive and scalable microservice systems.

6. DISCUSSION

6.1. Analysis of Results and Comparison with Existing Methods

The integration of CARGO into microservice partitioning has yielded promising results, particularly in enhancing partition quality and system performance. When compared to existing methods, CARGO demonstrates superior capabilities in reducing distributed transactions and improving operational metrics such as latency and throughput. This comparative analysis underscores the potential of AI-guided techniques in refining microservice architectures, offering a compelling case for the adoption of CARGO in migration strategies.

6.2. Potential Limitations and Areas for Improvement

Despite its advantages, the application of CARGO may encounter certain limitations. The effectiveness of CARGO is influenced by the accuracy of the system dependency graph; inaccuracies in this graph can lead to suboptimal partition refinements. Additionally, the computational overhead associated with the iterative refinement process may be a consideration in resource-constrained environments. Future research could focus on enhancing the scalability of CARGO, improving the precision of dependency modeling, and reducing computational demands to broaden its applicability.

6.3. Implications for Future Research and Practice

The promising outcomes associated with CARGO's application suggest several avenues for future exploration. Research could investigate the integration of CARGO with other AI-driven techniques to further enhance partitioning strategies. Practically, organizations undertaking the migration to microservices could benefit from adopting CARGO as a tool to streamline the decomposition process, optimize system performance, and achieve a more efficient transition from monolithic architectures. Continued advancements in AI and machine learning are likely to further augment the capabilities of tools like CARGO, paving the way for more sophisticated and effective migration solutions.

7. CONCLUSION

7.1. Summary of Findings

The integration of Artificial Intelligence (AI) into the migration of monolithic applications to microservices architectures has demonstrated significant advancements in both the efficiency and effectiveness of the process. The application of AI-driven techniques, such as the Context-sensitive lAbel pRopaGatiOn (CARGO) method, has led to substantial improvements in partition quality. Specifically, studies have shown that employing CARGO can reduce distributed transactions and enhance system performance, with observed reductions in latency by 11% and increases in throughput by 120% on average. These findings underscore the transformative potential of AI in modernizing legacy systems, facilitating a smoother transition to scalable and efficient microservice architectures.

7.2. Contribution to the Field of AI-Driven Application Migration

The exploration and implementation of AI-driven methodologies have significantly enriched the field of application migration. By automating complex processes such as code analysis, dependency mapping, and partition refinement, AI contributes to reducing manual intervention, minimizing errors, and accelerating migration timelines. For instance, Google's adoption of large language models (LLMs) for internal code migrations has led to notable reductions in migration timeframes, highlighting the efficacy of AI in handling intricate migration tasks. Furthermore, AI's capability to predict potential migration risks and optimize resource allocation enhances the reliability and success rates of migration projects, thereby advancing the state-of-the-art in the field.

7.3. Future Directions for Enhancing Migration Techniques

Looking ahead, the continuous evolution of AI presents promising avenues for further enhancing migration techniques. Future research is likely to focus on developing adaptive AI models capable of seamlessly integrating with various cloud-native architectures, facilitating hybrid and multi-cloud migrations. Such advancements would offer greater flexibility and scalability, addressing the diverse needs of modern enterprises. Additionally, the integration of generative AI holds potential in automating data generation and validation processes, ensuring data integrity and completeness during migrations. As AI technologies mature, their application in migration is expected to become more sophisticated, incorporating predictive analytics and real-time monitoring to proactively address challenges and optimize migration strategies. These developments are poised to further transform the migration landscape, offering organizations more efficient, secure, and cost-effective pathways to modernize their applications.

REFERENCES

[1] Nitin, V., Asthana, S., Ray, B., & Krishna, R. (2022). CARGO: AI-Guided Dependency Analysis for Migrating Monolithic Applications to Microservices Architecture. *arXiv preprint arXiv:2207.11784*.
[2] Nikolov, S., Codecasa, D., Sjovall, A., Tabachnyk, M., Chandra, S., Taneja, S., & Ziftci, C. (2025). How is Google using AI for internal code migrations? *arXiv preprint arXiv:2501.06972*.
[3] Kumar, A. (2024). AI-Driven Innovations in Modern Cloud Computing. *arXiv preprint arXiv:2410.15960*.
[4] Ramachandran, A. (2024). Harnessing Advanced Artificial Intelligence for Enhanced Enterprise Data Migration: A Comprehensive Analysis. *ResearchGate*.
[5] Vaidya, A., Vankayalapati, M. K., Chan, J., Ibraimoski, S., & Moran, S. (2024). A Generative AI Assistant to Accelerate Cloud Migration. *arXiv preprint arXiv:2401.01753*.
[6] AI-Driven Migration Testing: 7 Best Practices. (2024). *Laminar*.
[7] Harnessing AI for Application Modernization. (2024). *CI Global*.
[8] Robinson, et al. (2022). Adaptive AI Models for Automating Legacy System Migration in Enterprise Environments. *ResearchGate*.
[9] AI-Driven Innovations in Modern Cloud Computing. (2024). *arXiv preprint arXiv:2410.15960*.

[10] How AI is Revolutionizing Cloud Migration. (2024). *Ampcus*.
[11] The Future of Content Migration with AI. (2024). *Texter AI*.
[12] Generative AI – Driving the Next Innovation in Data Migration. (2024). *Onix*.
[13] AI-Driven Innovations in Modern Cloud Computing. (2024). *arXiv preprint arXiv:2410.15960*.
[14] AI-Driven Migration Testing: 7 Best Practices. (2024). *Laminar*.
[15] Harnessing AI for Application Modernization. (2024). *CI Global*.
[16] SUNKARA, S. K. (2025). LEVERAGING AI, IoT, AND BLOCKCHAIN FOR SCALABLE DIGITAL TRANSFORMATION IN POST-HARVEST SUPPLY CHAINS: A MULTI-SECTOR APPROACH TO ENHANCING EFFICIENCY AND TRACEABILITY (Vol. 26, Issue 7, pp. 2757–2766).