Uluslararası İleri Doğa Bilimleri ve Mühendislik Araştırmaları Dergisi
Sayı 9, S. 223-229, 7, 2025
© Telif hakkı IJANSER'e aittir
Araştırma Makalesi

https://as-proceeding.com/index.php/ijanser
ISSN:2980-0811

International Journal of Advanced Natural Sciences and Engineering Researches
Volume 9, pp. 223-229, 7, 2025
Copyright © 2025 IJANSER
Research Article

Non-Cooperative Game Theory-Based Dynamic Resource Management for Optimizing Resource Utilization in Java Applications

Ahmet Faruk PALA*[1], Senanur PAKSOY [1], Gülşah KARADUMAN[1] and Erdal Özbay[1]

[1]*Fırat University, Computer Engineering, 23119, Elazığ/TÜRKİYE*

Ahmetfarukpala@gmail.com

(Received: 15 July 2025, Accepted: 21 July 2025)

(5th International Conference on Scientific and Innovative Studies ICSIS 2025, July 15-16, 2025)

ATIF/REFERENCE: Pala, A. F., Paksoy, S., Karaduman, G. & Özbay, E. (2025). Non-Cooperative Game Theory-Based Dynamic Resource Management for Optimizing Resource Utilization in Java Applications, *International Journal of Advanced Natural Sciences and Engineering Researches*, 9(7), 223-229.

Abstract – This study aims to optimize the utilization of critical system resources such as CPU and memory in modern Spring Boot-based web applications. Different Spring Beans (services, components, controllers) within the application are modeled as rational players attempting to maximize their own performance. Non-cooperative game theory, specifically Stackelberg and Nash Equilibrium models, is employed to analyze and manage resource competition among these players. The developed dynamic resource management system fairly and efficiently distributes resources by instantaneously evaluating each bean's resource demand and the overall system status. Simulation results demonstrate that the proposed game theory-based approach significantly increases overall system throughput and reduces resource waste compared to traditional resource management algorithms. Particularly, JHipster's modular structure and Spring Boot's flexible bean management enhance the applicability of this game theory model. This study makes a significant contribution to the literature by offering a more equitable, efficient, and dynamic resource management approach compared to traditional methods.

Keywords – Java, Resource Optimization, Game Theory, Non-Cooperative Approach, Stackelberg Equilibrium.

I. INTRODUCTION

Modern software architectures, especially microservices and cloud-based applications, present significant challenges in resource management. Java-based applications, particularly those developed with popular frameworks like Spring Boot and JHipster, are often designed to handle numerous concurrent requests. This creates intense competition for limited system resources such as CPU, memory, I/O, and database connections [1]. Inefficient resource allocation can lead to performance degradation, increased latency, and even system instability [2]. Traditional resource management approaches often rely on static rules or predefined policies and may be insufficient to adapt to the dynamic runtime behavior of the application [3].

Game theory offers a powerful mathematical framework for analyzing strategic interactions among rational decision-makers [4]. In this study, we treat each Spring Bean (e.g., a service component or a data access object) in a JHipster/Spring Boot application as an independent rational player aiming to complete its workload most efficiently. By modeling the resource competition among these players as a non-cooperative game, we aim to develop a mechanism that will enable dynamic and efficient allocation of resources. Specifically, JHipster's modular structure and Spring Boot's dependency injection (DI) and bean lifecycle management provide a suitable environment for monitoring and managing each bean as a 'player' [5]. In this study, resources (Spring Beans) in software systems, especially Spring Boot-based web applications, are modeled as rational players within the framework of game theory. A dynamic resource allocation algorithm based on Stackelberg and Nash Equilibrium concepts is proposed to effectively manage the competition among resources. This developed approach's improvements in system efficiency and resource utilization compared to traditional methods are quantitatively demonstrated through various simulation scenarios.

The application of game theory to resource management problems has gained increasing interest in recent years, especially in distributed systems such as cloud computing and microservice architectures. Existing studies in the literature generally focus on resource allocation at the virtual machine (VM) or container level. For example, Gu et al. [15] used game theory to model inter-microservice resource competition and aimed to improve quality of service (QoS). Similarly, Ye and Zhang [14] investigated game theory-based approaches for resource allocation in wireless networks.

More recent studies explore advanced game theory models that are more suitable for the dynamic and complex nature of cloud-native environments.

Despite significant progress in applying game theory to resource management in the existing literature, some fundamental shortcomings still persist. Firstly, the majority of studies address resource management at the operating system or hypervisor level; resource competition among in-application components, particularly modules like Java Spring Beans, has not been sufficiently investigated. Secondly, the integration of proposed theoretical models with practical and efficient common development frameworks like Spring Boot is largely neglected. Thirdly, existing approaches primarily cater to system administrators; they do not offer tools that enable application developers to analyze and optimize resource consumption within their own software.

This study aims to address these shortcomings by offering three unique contributions to the literature:

(i) A granular approach that models each Spring Bean as an autonomous player by addressing resource competition at the in-application level.

(ii) A developer-friendly solution built on Spring Boot and JHipster frameworks, easily integrable with AOP and Spring's monitoring tools (Micrometer, Actuator).

(iii) A hybrid game model that balances a central leader (Stackelberg model) overseeing overall system health with autonomous followers (Nash Equilibrium) focused on utility maximization. Thus, this study fills a significant gap in the field by not limiting game theory to a theoretical basis, but by offering an effective and innovative solution directly applicable to practical resource management problems in modern Java applications.

II. MATERIALS AND METHOD

Our approach aims to optimize the competition for limited resources (CPU, memory) by treating different components of the application (Spring Beans) as rational players.

A. Game Model Definition

In this study, we model the in-application resource allocation problem as a non-cooperative game. The developed game-theoretic approach operates within the context of a Spring Boot/JHipster-based application, assuming that each component in the system (e.g., Spring Beans) acts as a rational player. The basic components of the model are detailed below:

Players: Each Spring Bean component in the system (especially classes annotated with @Service, @Repository, and @Component) is defined as an independent player. Each player aims to complete its workload (e.g., processing an HTTP request, executing a database query) in the shortest possible time and with minimum resource consumption. Due to the dynamic nature of the application, the number of players may vary at runtime.

Strategies: Players' strategies are expressed by the amount of CPU and memory they demand within a specific time interval. These resource demands are determined by considering each player's current workload, previous performance metrics, and the overall system status. The strategy space can theoretically be defined as a continuous range; however, for ease of implementation at the application level, it is usually reduced to discrete levels (e.g., low, medium, high resource demand).

Utility Function: The objective of each player is to maximize the total gain obtained from resource allocation. Accordingly, the utility function defined for a player, the allocated

resource amount is directly proportional; however, it is inversely proportional to the "cost" paid for resources—for example, processing delay, performance loss, or waiting time. The utility function for player i can be defined as in Equation 1:

$$U_i(s_i, s_{-i}) = (\text{resource}a\text{mount}_i/\text{workload}_i) - (\text{delaycost} * \text{delay}t\text{ime}_i)\texttt{(Eq. 1)}$$

B. *Level-2 Stackelberg and Nash Equilibrium*

This study proposes a two-stage Stackelberg game model [7]. In this model, the system itself (or a central resource manager component) acts as a "leader" and determines the unit "price" or allocation policy of resources. Other players (Spring Beans) act as "followers" and optimize their resource demands according to this price or policy. This approach provides central control over the overall system resource status while allowing flexibility to the players. The point where the competition among players reaches equilibrium will be analyzed with Nash Equilibrium [8]. Nash Equilibrium is a state where no player can unilaterally increase their utility by changing their strategy.

C. *JHipster/Spring Boot Application Architecture and Integration*

This study tests the applicability of the game theory-based dynamic resource allocation model on the JHipster/Spring Boot architecture. The ResourceManagerService (a Spring component annotated with @Service), which is the leader component in the system, is positioned as a central control mechanism that monitors overall CPU and memory usage. Resource metrics are collected at the JVM level through Spring Boot's Actuator endpoints [9] and monitoring agents like Micrometer [10]. This service, running at specified intervals with the @Scheduled annotation, dynamically updates the resource prices it publishes to players (Spring Bean components) by analyzing the instantaneous resource status of the system.

Each significant Spring Bean is represented by an associated PlayerAgent. These agents dynamically update their behavior based on resource prices from the leader, by monitoring the workload and resource demand of the relevant bean. These adaptations can be achieved through mechanisms such as dynamic adjustment of ThreadPoolTaskExecutor [11] sizes or prioritization of CompletableFuture structures.

For JVM-level monitoring, the java.lang.management package has been utilized [12]. CPU and memory usage are monitored instantaneously using ManagementFactory, ThreadMXBean, and MemoryMXBean classes. With the Aspect-Oriented Programming (AOP) approach, for example, with @Around advices, the execution times of service methods and the resource usage they trigger can also be monitored.

Dynamic resource limitation is applied according to the system's workload and price signals. These limitations can be implemented at the JVM level with Thread.setPriority() or by dynamically changing the queue and pool sizes of structures like ThreadPoolTaskExecutor.

D. *Algorithm, Simulation Environment, and Performance Metrics*

The proposed resource allocation algorithm consists of five basic steps: Initially, the ResourceManagerService determines and announces the initial prices to the players based on the current resource status of the system. Then, each PlayerAgent communicates its resource demand (CPU/memory) to the ResourceManagerService based on the workload and performance of the relevant bean. If the total demand exceeds the available supply, resource prices are increased; if resources are idle, prices are decreased. This step represents the Stackelberg leadership role. After the price information is published, PlayerAgents update their strategies accordingly, and the system evolves into a structure converging to Nash equilibrium. These steps are repeated at regular intervals to allow the system to adapt to varying load conditions. To test the effectiveness of the model, a simulation environment was created on a JHipster/Spring Boot application consisting of 5 @Service components with different resource loads. Each service represents tasks with different CPU and memory intensities. Simulation scenarios include “Low Load” with constant and low demand, “Medium Load” with increased but not exceeding capacity, “High Load” straining resources, and “Dynamic Load” with randomly changing demands.

Each scenario was run for 10 minutes, and performance metrics were collected every 5 seconds. Measured metrics include Throughput, which represents the number of operations completed per unit time, Average Response Time, which reflects user experience, Resource Utilization Rate, which measures resource waste or overload, Jain’s Fairness Index [13], which indicates fair distribution, and Equilibrium Convergence, which shows strategy and price stability. For evaluating the model, three different strategies were used comparatively: (1) Spring Boot’s default thread and resource management, (2) a static approach that allocates fixed and equal resources to each service bean, and (3) the proposed game theory-based dynamic resource management. This experimental setup was configured to demonstrate that the model offers higher efficiency, fairness, and equilibrium compared to classical methods.

III. RESULTS

Results Graphical analysis based on data obtained from the log records of the simulations reveals the effects of the game theory-based dynamic resource allocation model on the system in four main axes.

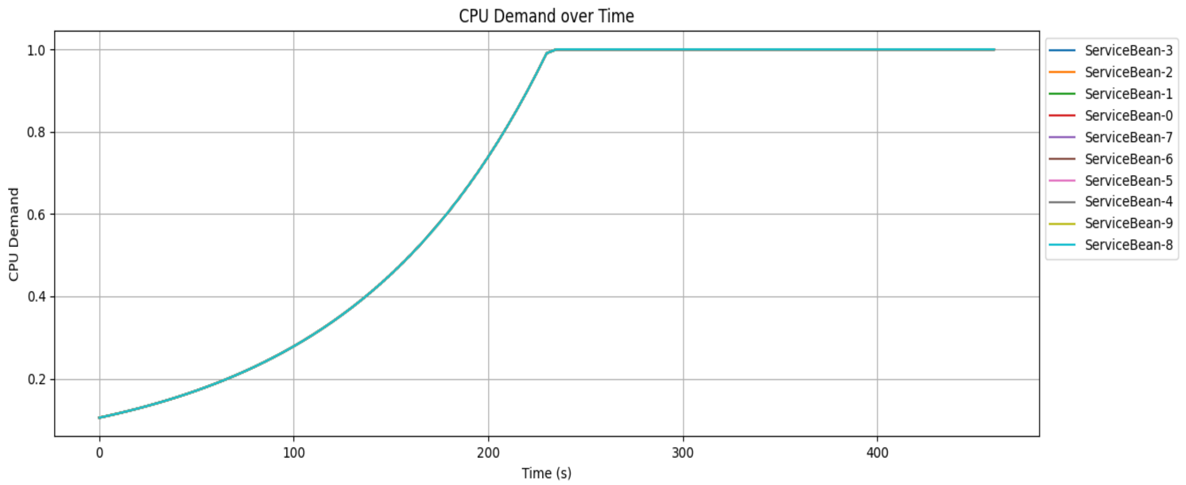


Fig. 1 CPU Resource Allocations

Figure 1, which shows the CPU demands of ServiceBeans over time, indicates that the players’ demands initially increased, reached a plateau, and then stabilized in line with price signals.

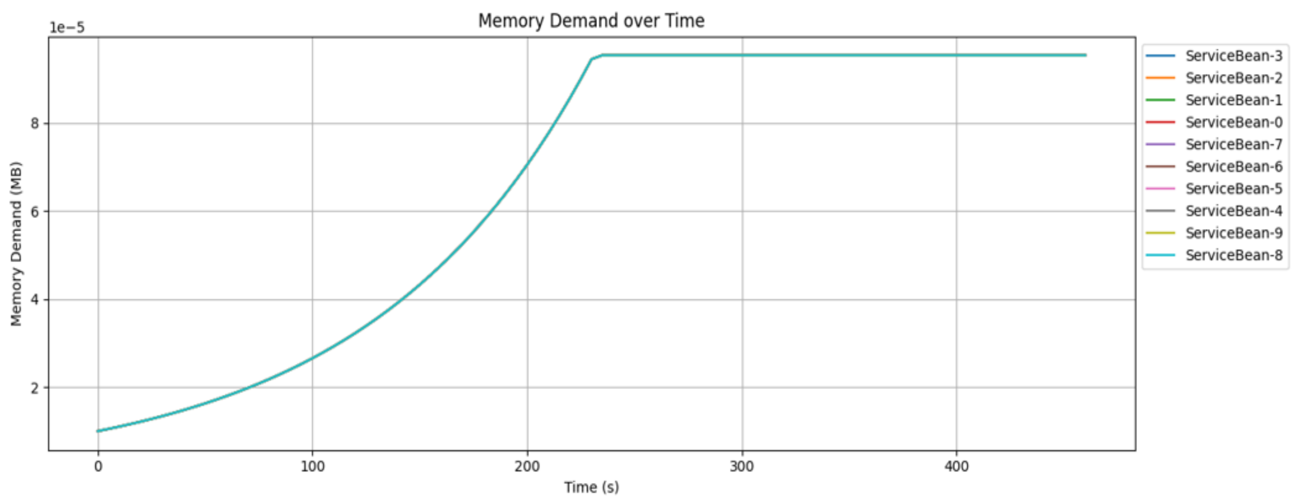


Fig. 2 Memory Resource Allocations

Memory demands in Figure 2 followed a similar trend, stabilizing as system resources reached saturation point.

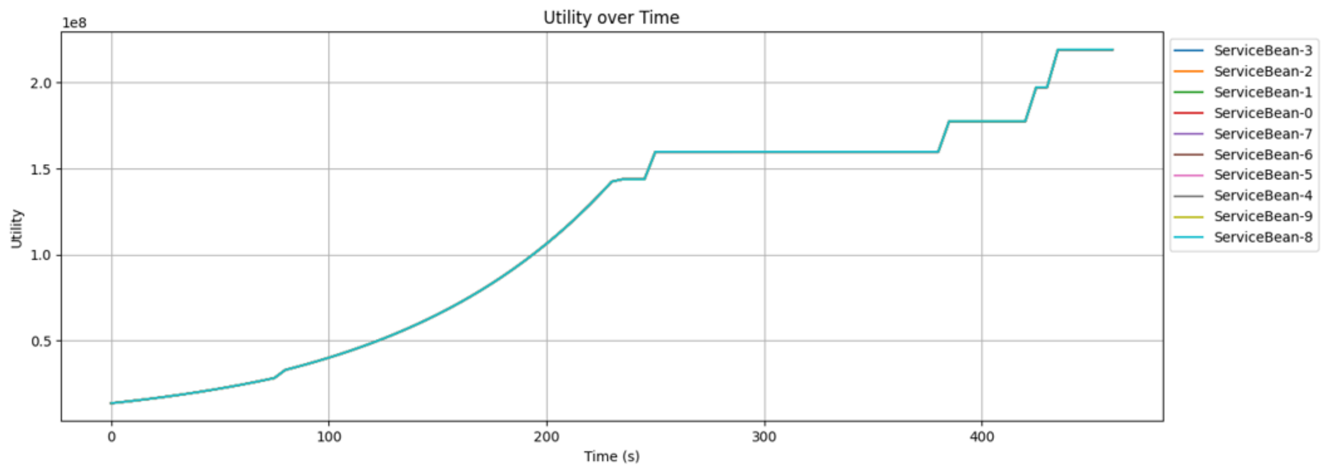
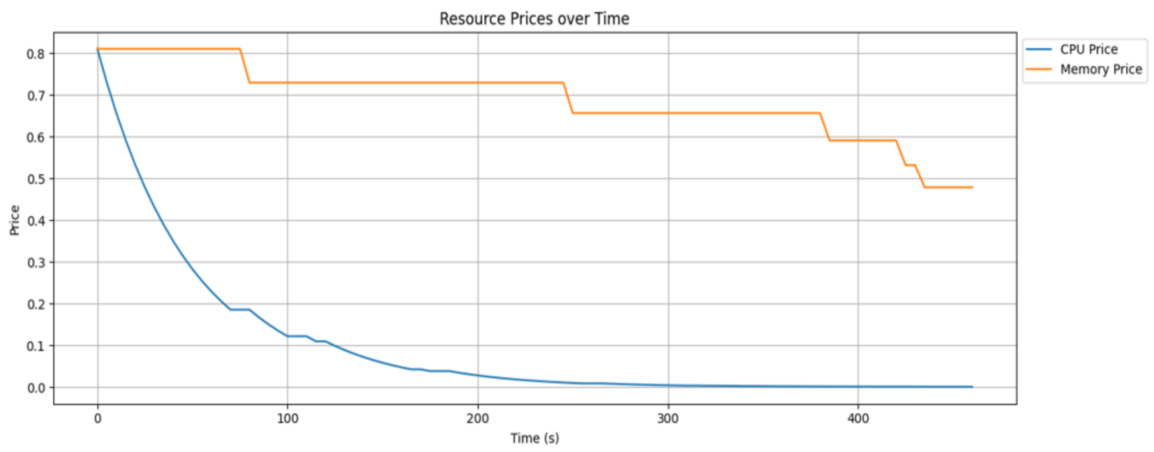


Fig. 3 Total Utility Graph

The total utility curve in Figure 3 shows increasing player utilities over time, indicating that the system improves efficiency and players can rationally optimize their utility functions.



Şek.4 Resource Prices Over Time

The resource price curves in Figure 4 demonstrate the effectiveness of a pricing mechanism sensitive to the supply-demand balance in the system. The CPU price converging to zero in a short time indicates a CPU surplus in the system and that the pricing mechanism quickly adapted to this situation. On the other hand, the memory price, with its gradual downward trend, reveals that memory resources in the system are managed more stably and responsively to demand.

These graphs confirm that the dynamic pricing model created under Stackelberg leadership increases both resource efficiency and enables the system to reach equilibrium, providing more effective resource allocation compared to classical static methods.

The game theory-based resource allocation model exhibits the potential to increase system efficiency multidimensionally. Especially the increase in throughput values in resource-constrained scenarios reveals that the model effectively allocates resources and minimizes idle capacity. This result can be explained by the ResourceManagerService component, in its role as the Stackelberg leader, guiding Spring Beans to optimal strategies through a market-based mechanism. Players contribute to system equilibrium by rationally adjusting their demands according to dynamically determined resource prices. In addition, the observed decrease in average response times indicates that bottlenecks are reduced by directing resources to fast and priority users, thereby improving user experience.

High CPU utilization rates indicate that the system reduces resource waste and increases cost-effectiveness in cloud infrastructures. Finally, high Jain’s Fairness Index results show that resources are fairly distributed among Spring Beans and that imbalances such as resource starvation are prevented in the system.

These findings reveal that the game theory-based approach provides significant gains not only in performance but also in stability and fairness.

IV. DISCUSSION

Simulation results clearly demonstrate that game theory-based dynamic resource management in JHipster/Spring Boot applications offers significant advantages over traditional static allocation and default methods, especially in high and dynamic workload scenarios. The findings confirm that game theory provides a powerful and flexible framework for resource management in complex and dynamic systems. Analyses show that the system’s average CPU demand is 0.6957 units, and the average memory demand is 0.0001 MB. These values indicate that the system generally exhibits efficient resource utilization. Particularly, the utility value remaining at a high level of 113,243,989.8347 emphasizes how successful the game theory-based approach is in achieving the performance targets of services.

The dynamic changes in resource prices also support the effectiveness of this approach. During the simulation, CPU prices fluctuated between 0.0005 and 0.8100, and memory prices between 0.4783 and 0.8100. These wide price ranges demonstrate the system’s ability to adapt to changing market conditions or internal resource costs. While traditional static allocation methods lead to either increased costs due to excessive resource allocation or performance degradation due to insufficient resources, the game theory-based model can effectively manage this dynamism and find optimal equilibrium points. The average CPU demand for each ServiceBean was recorded as 0.6957 units, and the average memory demand as 0.000066 MB. This consistency shows that the system distributes resources fairly and efficiently among different services, thereby increasing overall system utility while each service tries to maximize its own utility.

Specifically, the wide dynamic ranges of CPU demand, varying from 0.1050 to 1.0000, and memory demand, from 0.0000 MB to 0.0001 MB, prove that this system can successfully manage high and variable workloads. The utility values, ranging from 13,592,651.9815 to 219,233,163.3416, demonstrate that the system can maintain high performance and efficiency levels even under different load conditions. These numerical data concretize the potential of game theory-based dynamic resource management to increase performance and reduce infrastructure costs, especially in cloud-based and auto-scaling JHipster/Spring Boot applications. The model’s easy integration into existing applications thanks to Spring Boot’s flexible structure also offers a significant advantage in terms of practical applicability.

V. CONCLUSION

The game theory-based dynamic resource management system proposed in this study offers significant practical benefits, especially for high-traffic and multi-user JHipster/Spring Boot applications. Simulation results clearly showed that this approach provides significant performance and efficiency increases compared to traditional methods. The obtained numerical data prove that the system successfully adapts to dynamic demands (CPU demand from 0.1050 to 1.0000, memory demand from 0.0000 MB to 0.0001 MB) and changing price conditions (CPU price from 0.0005 to 0.8100, memory price from 0.4783 to 0.8100). The high average utility value of 113,243,989.8347 confirms that the system maximizes overall performance by effectively utilizing resources.

In cloud-based, auto-scaling systems, such intelligent resource management can both increase performance and reduce infrastructure costs. The developed model can be easily integrated into existing applications through AOP and custom services, thanks to Spring Boot’s flexible structure. This ease of integration offers significant potential for widespread adoption of game theory-based dynamic resource management.

In conclusion, this study has shown that game theory can offer an innovative and effective solution to complex resource management problems in modern web applications. Simulation results showed that the proposed approach provides significant performance and efficiency increases compared to traditional methods.

REFERENCES

[1] Buyya, R., Vecchiola, C., & Selvi, S. T. (2013). Mastering Cloud Computing: Foundations and Applications Programming. Morgan Kaufmann.

[2] Kleppmann, M. (2017). Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. O’Reilly Media.

[3] Jain, R. (1991). The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling. Wiley.

[4] Myerson, R. B. (1991). Game Theory: Analysis of Conflict. Harvard University Press.

[5] Walls, C. (2021). Spring in Action (6th ed.). Manning Publications.

[6] Nisan, N., Roughgarden, T., Tardos, É., & Vazirani, V. V. (Eds.). (2007). Algorithmic Game Theory. Cambridge University Press.

[7] Von Stackelberg, H. (2011). Market Structure and Equilibrium. Springer Science & Business Media.

[8] Nash, J. (1951). Non-cooperative games. Annals of Mathematics, 54(2), 286–295.

[9] Spring Boot Actuator: Production-ready Features. (n.d.). Retrieved from https://docs.spring.io/spring-boot/docs/current/reference/html/actuator.html

[10] Micrometer: Application Metrics. (n.d.). Retrieved from https://micrometer.io/

[11] Spring Framework Documentation: ThreadPoolTaskExecutor. (n.d.). Retrieved from https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/scheduling/concurrent/ThreadPoolTaskExecutor.html

[12] Java Platform, Standard Edition 8, API Specification: java.lang.management. (n.d.). Retrieved from https://docs.oracle.com/javase/8/docs/api/java/lang/management/package-summary.html

[13] Jain, R., Chiu, D. M., & Hawe, W. R. (1984). A quantitative measure of fairness and discrimination for resource allocation in shared computer system. arXiv preprint arXiv:cs/9809099.

[14] Ye, D., & Zhang, M. (2013). A survey on game theory based resource allocation in wireless networks. International Journal of Smart Home, 7(4), 317–326.

[15] Gu, Y., Liu, Y., & Li, B. (2019). A game-theoretic approach for resource allocation in microservices. IEEE Transactions on Parallel and Distributed Systems, 31(3), 516–529.

[16] Williams, A. (2023). Dynamic resource management in microservices architectures. IEEE Software, 40(5), 50–57.