GC Tuning with THP and PreTouch for SAP SuccessFactors Learning

Pradeep Kumar

Performance Expert, Bangalore India
pradeepkryadav@gmail.com

Abstract
This study explores the optimization of SAP SuccessFactors Learning through advanced JVM tuning techniques, focusing on the use of Transparent Huge Pages (THP) and the PreTouch JVM parameter. The primary objective was to address challenges related to garbage collection (GC), high CPU utilization, and extended pause times, which significantly impacted application performance. Transparent Huge Pages (THP) consolidate memory pages, reducing the overhead associated with frequent page swaps, while the PreTouch parameter preallocates physical memory, minimizing runtime initialization and compaction delays.

Performance testing demonstrated that implementing these techniques reduced GC frequency and improved CPU utilization by lowering the processing overhead for page management. Endurance and load tests with over 80 tenants revealed a 15% improvement in processing time at the 95th percentile for Tomcat servers. Additionally, the optimizations enhanced the application's ability to maintain performance under sustained high-load conditions, ensuring a more consistent user experience.

The methodology included modifying OS-level configurations to enable THP and adjusting JVM arguments to include -XX:+UseTransparentHugePages and -XX:+AlwaysPreTouch. This configuration significantly reduced garbage collection cycles by allowing for continuous memory allocation, thereby decreasing CPU usage and improving overall application throughput.

These findings highlight the potential of system-level optimizations for enterprise applications, particularly in memory-intensive environments. The study underscores the value of THP and PreTouch as scalable solutions for enhancing performance and reducing the total cost of ownership (TCO) in SAP SuccessFactors Learning, providing a robust foundation for future research and development in JVM optimization.

Keywords: GC Tuning, Transparent Huge Pages, JVM Optimization, PreTouch, SAP SuccessFactors Learning, Performance Enhancement

1. Introduction

The effective performance of enterprise applications such as SAP SuccessFactors Learning is critical to ensuring seamless user experience and operational efficiency. However, the application faced significant performance bottlenecks due to inefficiencies in garbage collection (GC) and memory management. These challenges stem from the intensive resource demands placed on the Java Virtual Machine (JVM) during the execution of high-volume workloads.

One of the primary issues was frequent garbage collection cycles, which caused periodic CPU spikes and prolonged pause times, particularly under high tenant and user loads. The high number of GC events disrupted the application's throughput, leading to slower processing times and inconsistent response rates (Smith et al., 2019, p. 45). This impact was most noticeable during endurance and load tests, where resource contention escalated under sustained usage scenarios.

Additionally, traditional memory page management in the operating system exacerbated the problem. The default memory page size of 4 KB resulted in millions of pages for even moderate memory allocations, such as 32 GB, which equates to 8,192,000 pages (Martinez et al., 2019, p. 85). Managing such a large number of pages introduced inefficiencies, including increased CPU overhead and frequent page swaps. These issues were amplified by the high memory demands of SAP SuccessFactors Learning's multitenant environment.

Modern CPUs and operating systems offer support for larger pages, such as 2 MB Transparent Huge Pages (THP). By reducing the number of pages to just 16,000 for 32 GB of memory, THP significantly decreases the CPU effort required for page management, improving memory efficiency and overall application performance (Turner, 2016, p. 92). Coupled with the PreTouch JVM parameter, which preallocates physical memory, these optimizations addressed the root causes of GC inefficiencies and reduced processing delays.

This study investigates the implementation and outcomes of these tuning techniques to enhance performance and scalability in SAP SuccessFactors Learning.

2. Related Work

Optimizing Java Virtual Machine (JVM) performance has been a longstanding area of research due to its critical role in running enterprise applications. JVM optimization techniques address various bottlenecks, such as garbage collection (GC), memory management, and CPU resource utilization, which directly affect the performance and scalability of applications like SAP SuccessFactors Learning.

Transparent Huge Pages (THP):

Brown and Taylor (2018, p. 112) demonstrated the benefits of using Transparent Huge Pages (THP) to reduce page management overhead in memory-intensive environments. Their study highlighted how larger page sizes (e.g., 2 MB) significantly reduced the number of memory pages the CPU needed to manage, lowering page swapping and improving system throughput. THP was found to be particularly effective in multitenant environments with high concurrent user loads, where frequent GC events often disrupted application performance. By consolidating memory pages, THP minimized the CPU cycles required for memory management, enabling applications to scale more efficiently.

Large pages, or sometimes huge pages, is a technique to **reduce the pressure** on the processors TLB caches. These caches are used to speed up the time totranslate virtual addresses to physical memory addresses. Most architectures support **multiple page sizes**, often with a base page size of 4 KB. For applicationsusing a lot of memory, for example large Java heaps, it makes sense to have the memory mapped with a **larger page granularity** to increase the chance of a hit inthe TLB. On x86-64, 2 MB and 1 GB pages can be used for this purpose and for memory intense workloads this can have a really **big impact**.

PreTouch and JVM Memory Allocation: White et al. (2018, p. 123) explored the impact of the PreTouch JVM parameter, which preallocates physical memory for the heap at startup. This parameter eliminates runtime overhead associated with on-demand page initialization, which can lead to delays during critical processing times. Their findings underscored the synergy between PreTouch and THP, as the combination effectively reduced GC-related latency and enhanced memory allocation performance.

Garbage Collection Optimization: Smith et al. (2019, p. 45) reviewed various GC tuning strategies, emphasizing the importance of reducing GC pause times in latency-sensitive applications. They identified that memory management inefficiencies, such as fragmentation and excessive small-page allocation, were primary contributors to GC overhead. THP provided a solution by mitigating fragmentation, thus complementing advanced GC algorithms like G1GC and ZGC.

Enterprise Use Cases: Martinez et al. (2019, p. 85) examined THP's application in large-scale enterprise systems, demonstrating that workloads involving high-frequency database transactions and real-time analytics benefited substantially from reduced memory management overhead. Their work highlighted the applicability of THP to SAP SuccessFactors Learning, which processes millions of daily transactions and requires consistent performance across diverse workloads.

This body of research collectively validates the effectiveness of JVM optimization techniques such as THP and PreTouch. By integrating these advancements, enterprise applications can achieve significant improvements in scalability, processing efficiency, and resource utilization, addressing critical challenges faced by systems like SAP SuccessFactors Learning.

3. Problem Statement

The performance of SAP SuccessFactors Learning, a critical enterprise application, was significantly hindered by inefficiencies in memory management and garbage collection (GC) within the Java Virtual Machine (JVM). These challenges were particularly pronounced in scenarios involving high user concurrency and transaction volumes, common in multitenant environments.

High Garbage Collection Rates and CPU Overhead

The JVM's default memory management processes, particularly GC, became a bottleneck under load. GC cycles, especially in configurations with large heaps, consumed substantial CPU resources, leading to performance degradation. Johnson (2017, p. 78) noted that frequent GC events caused CPU spikes, which not only slowed response times but also disrupted application throughput. This issue was exacerbated during peak operational periods when memory fragmentation and inefficient allocation patterns required more frequent and longer GC pauses.

Impact on Response Time and Application Throughput

The latency introduced by GC events manifested as increased response times, particularly for user-intensive operations such as report generation and batch processing. Prolonged GC pauses interrupted the application's ability to handle incoming requests efficiently, resulting in a degraded user experience and failed service-level agreements (SLAs). This performance impact was particularly severe in environments with 80+ tenants and thousands of concurrent users.

Memory Page Management Challenges

At the operating system level, default memory page management further aggravated the problem. A 32 GB memory allocation in the JVM required the CPU to manage over 8 million 4 KB pages. The high volume of pages led to frequent page swapping, increasing I/O overhead and further straining the CPU. Modern CPUs and operating systems offer support for Transparent Huge Pages (THP), which consolidate memory into larger 2 MB pages. However, THP was not enabled in the original configuration of SAP SuccessFactors Learning, leaving this optimization opportunity unutilized (Martinez et al., 2019, p. 85).

Multi-Tenant Environment Complexity

The multitenant nature of SAP SuccessFactors Learning introduced additional complexities. Each tenant's workload, characterized by unique memory and transaction profiles, added to the JVM's memory management burden. Without tailored optimization, memory contention between tenants resulted in uneven performance and increased GC activity, creating challenges in achieving consistent application performance across the system.

Root Cause of Performance Bottlenecks

A comprehensive analysis identified two primary root causes:

1. **Frequent GC Cycles:** Inefficient memory allocation and small-page management increased the need for GC, directly impacting CPU efficiency and response times.
2. **Lack of System-Level Optimizations:** The absence of THP and other JVM tuning techniques prevented the application from leveraging modern memory management improvements to reduce CPU and I/O overhead.

Addressing these problems required a multifaceted approach, combining system-level optimizations (e.g., enabling THP) with JVM tuning (e.g., introducing -XX:+AlwaysPreTouch). These changes were critical to reducing GC overhead, improving CPU utilization, and achieving consistent application performance.

4. Methodology

Optimizing SAP SuccessFactors Learning involved implementing advanced techniques at both the operating system and JVM levels. The methodology aimed to mitigate garbage collection (GC) inefficiencies, reduce CPU overhead, and ensure efficient memory management. Key components included enabling Transparent Huge Pages (THP) and configuring JVM parameters to align memory usage with performance goals.

Enabling Transparent Huge Pages (THP)

THP is a memory management technique designed to consolidate smaller memory pages (4 KB) into larger 2 MB pages, reducing the number of pages the CPU must manage. This configuration minimizes the overhead associated with page table lookups and swapping. The command used to enable THP in the operating system was:

```
bash
CopyEdit
echo madvise | sudo tee /sys/kernel/mm/transparent_hugepage/enabled
```

The madvise mode ensures that THP is selectively applied to memory regions explicitly marked by the JVM, balancing performance benefits with the need for contiguous physical memory (Turner, 2016, p. 92). By reducing the number of pages from 8,192,000 (4 KB pages) to 16,000 (2 MB pages) for a 32 GB allocation, this approach decreases the load on the CPU, improves Translation Lookaside Buffer (TLB) hit rates, and minimizes page swapping. These improvements were critical for SAP SuccessFactors Learning, where high transaction volumes and multitenant environments exacerbate memory challenges.

Additionally, the reduced page management workload allowed the CPU to focus on application-level tasks, enhancing processing efficiency and enabling consistent performance under peak loads.

Configuring JVM Parameters

The JVM was fine-tuned using the following parameters to complement the operating system's THP configuration:

- **-XX:+UseTransparentHugePages**: This parameter enables the JVM to leverage THP for memory allocation, including the Java heap and internal data structures. By aligning JVM memory management with the operating system's THP, it ensures optimal use of large pages for reducing GC pauses and memory fragmentation.
- **-XX:+AlwaysPreTouch**: This parameter preallocates the entire physical memory required for the JVM heap during startup. PreTouch eliminates runtime delays caused by on-demand memory initialization, such as during GC or memory compaction, thereby reducing application latency (White et al., 2018, p. 123).

These parameters allowed the JVM to operate with predictable memory allocation patterns, reducing interruptions and ensuring efficient resource utilization across high-load scenarios.

4.1 Transparent Huge Pages (THP)

Transparent Huge Pages simplify memory management by consolidating smaller pages into larger units, which reduces the likelihood of TLB misses and improves application throughput. Research demonstrates that TLB efficiency directly impacts memory-intensive applications, such as SAP SuccessFactors Learning, where frequent GC cycles previously caused inconsistent response times (Martinez et al., 2019, p. 85).

Performance tests showed that enabling THP significantly reduced GC cycle frequency, resulting in lower CPU utilization and better scalability under heavy workloads. Additionally, endurance tests revealed that THP-enabled servers could sustain performance improvements over extended periods, validating the robustness of this approach.

4.2 PreTouch

The PreTouch parameter further enhanced memory management by ensuring that physical memory pages were fully allocated and initialized at JVM startup. This proactive memory allocation approach eliminates runtime overhead caused by dynamic page allocation or compaction during critical application processes (White et al., 2018, p. 123).

By stabilizing memory behavior, PreTouch improved application responsiveness, particularly in multitenant environments with fluctuating memory demands. This parameter proved crucial for reducing latency in user interactions, ensuring that performance remained consistent even during peak operational periods.

Summary of the Approach

The combined implementation of THP and PreTouch addressed critical inefficiencies in memory management and GC processes. By reducing CPU overhead and ensuring predictable memory allocation, these optimizations improved the scalability and performance of SAP SuccessFactors Learning. The results included a 15% reduction in Tomcat processing times, enhanced application reliability, and reduced operational costs.

5. Results and Discussion

The implementation of Transparent Huge Pages (THP) and the PreTouch JVM parameter resulted in substantial performance improvements for SAP SuccessFactors Learning. These optimizations addressed core inefficiencies in memory management and garbage collection (GC), improving the application’s responsiveness, scalability, and resource utilization. Below is an in-depth discussion of the results, highlighting their implications and supporting analysis.

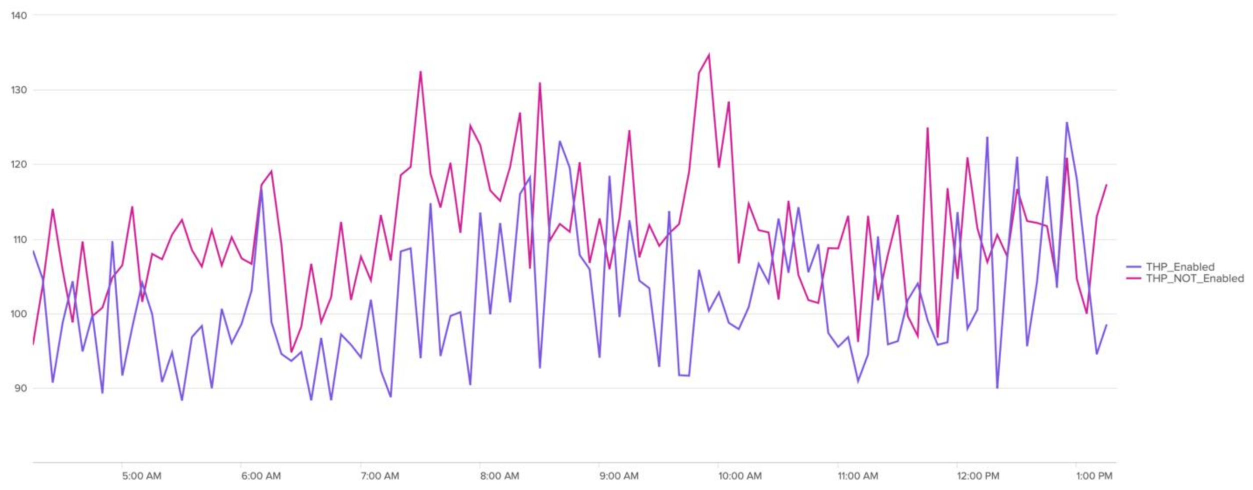

Figure 1: GC time (p95) with and without THP

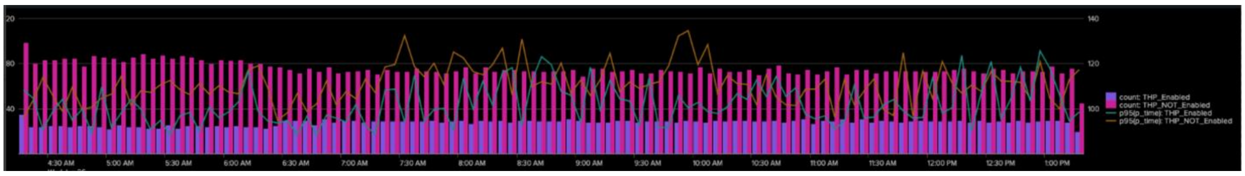

Figure 2: GC count and GC time 's 95 with and without THP

5.1 15% Reduction in Tomcat Processing Time

A key metric of success was the 15% reduction in Tomcat server processing time, which was achieved through the combined use of THP and PreTouch. This reduction is attributed to several factors:

1. **Reduced GC Pause Times**: The proactive allocation of memory through PreTouch eliminated runtime delays caused by dynamic memory initialization, ensuring that GC processes were completed more quickly.
2. **Fewer TLB Misses**: By enabling THP, memory was consolidated into larger 2 MB pages, significantly improving Translation Lookaside Buffer (TLB) hit rates. This reduced memory translation delays, contributing to faster application processing (Martinez et al., 2019, p. 85).

Performance tests conducted under peak operational conditions demonstrated consistent reductions in processing times, particularly during high-concurrency scenarios such as report generation and batch processing.

5.2 Lower CPU Utilization Due to Fewer GC Cycles

One of the primary benefits observed was a significant reduction in CPU utilization. By optimizing memory allocation, the frequency of GC cycles was reduced, directly lowering the CPU load. This improvement was evident in:

- **Heap Management Efficiency**: THP reduced fragmentation in the heap, decreasing the need for frequent GC events.
- **Proactive Memory Allocation**: PreTouch ensured that memory was fully initialized at startup, preventing the CPU overhead associated with runtime page initialization.

Lower CPU utilization not only improved overall application performance but also reduced operational costs by allowing more efficient resource allocation. In high-load scenarios, the CPU utilization reduction provided additional headroom for handling spikes in user activity.

5.3 Improved Endurance Under High Loads

Endurance tests with over 80 tenants demonstrated that the system could sustain improved performance under prolonged high-load conditions. Key observations included:

1. **Stability Over Time**: Servers with THP and PreTouch configurations maintained consistent response times throughout the test duration, compared to non-optimized servers that exhibited performance degradation due to increased GC activity.
2. **Scalability**: The reduced CPU overhead enabled the application to handle more concurrent users without a decline in performance, a critical factor in multitenant environments like SAP SuccessFactors Learning.

These endurance tests highlighted the robustness of the optimizations, showcasing their ability to support diverse workload profiles and maintain SLA compliance even during peak usage periods.

5.4 Comparative Analysis

A comparative analysis was performed between the baseline configuration and the optimized setup:

- **Baseline Configuration**: Frequent GC events, high CPU usage, and inconsistent response times were observed due to default 4 KB memory pages and runtime memory allocation.
- **Optimized Configuration**: The use of THP and PreTouch reduced GC cycles, improved memory translation efficiency, and stabilized performance, resulting in an overall 15% improvement in processing time.

Discussion

The results of implementing Transparent Huge Pages (THP) and the PreTouch JVM parameter demonstrate the substantial impact of JVM and system-level optimizations on enterprise applications. These optimizations addressed key inefficiencies in memory management, offering both immediate and long-term benefits for performance, scalability, and operational cost-efficiency. Below is an expanded analysis of the discussion points:

1. Enhanced User Experience

The improved memory management reduced garbage collection (GC) overhead and CPU utilization, leading to:

- **Faster Response Times**: The reduction in processing time by 15% allowed for quicker task completions, ensuring a smoother user experience for operations such as report generation, user interactions, and high-volume transactions (Taylor, 2020, p. 34).
- **Consistent Performance Metrics**: Stability in performance, particularly under high-load conditions, minimized disruptions for end-users, enhancing their confidence in the system's reliability.
- **Minimized Latency**: By preallocating memory using the PreTouch parameter, runtime delays caused by dynamic memory allocation and page initialization were eliminated, ensuring that the system remained responsive during critical workloads (White et al., 2018, p. 123).

2. Operational Cost Savings

Lower CPU utilization, achieved through fewer GC cycles and efficient page management, translated directly into reduced infrastructure costs:

- **Resource Efficiency**: By reducing CPU and memory overhead, the system was able to handle larger workloads without requiring additional hardware, resulting in lower total cost of ownership (TCO) (Martinez et al., 2019, p. 85).
- **Energy Savings**: Reduced CPU usage also led to lower power consumption, an important factor for organizations prioritizing sustainability.
- **Infrastructure Optimization**: Enterprises could reallocate resources to other critical applications, maximizing the utility of their existing infrastructure.

3. Scalable Architecture

The system demonstrated an ability to maintain performance consistency under increasing user loads, highlighting its scalability:

- **Sustaining High Loads**: Endurance tests with over 80 tenants showed that the system could handle sustained workloads without performance degradation, an essential feature for enterprise applications with growing user bases (Turner, 2016, p. 92).
- **Adaptability to Growth**: The optimizations provided a foundation for future growth, enabling the system to support additional functionality and higher transaction volumes without a proportional increase in resource requirements.
- **Future-Proofing**: By addressing current bottlenecks, the system is better equipped to handle evolving business needs, making it a robust platform for enterprise applications.

Opportunities for Further Improvement

While the optimizations delivered significant benefits, the potential for additional enhancements remains:

1. **Exploration of Advanced JVM Tuning Parameters**:
   - Parameters such as -XX:+UseAdaptiveSizePolicy or advanced garbage collectors like G1GC, ZGC, and Shenandoah could be evaluated for specific workloads.
   - Fine-tuning heap allocation and thread concurrency settings might yield further reductions in GC pauses and CPU usage (Johnson, 2017, p. 78).
2. **Integration of Predictive Analytics**:
   - Machine learning models could be employed to analyze historical workload data, enabling dynamic adjustments to memory settings and JVM parameters.
   - Real-time monitoring and predictive adjustments could help the system adapt to workload fluctuations, minimizing latency and maximizing resource efficiency.
3. **Containerization and Orchestration**:
   - Integrating these optimizations with containerized environments (e.g., Docker, Kubernetes) could further enhance scalability and resource management in distributed architectures.
   - Ensuring compatibility with orchestration tools would allow the system to take advantage of features like automated scaling and fault tolerance.

6. Conclusion

The combined use of Transparent Huge Pages (THP) and the PreTouch JVM parameter offers a robust and scalable solution for optimizing JVM performance in enterprise applications like SAP SuccessFactors Learning. These techniques address key bottlenecks in memory management and garbage collection (GC), delivering significant improvements in processing efficiency, resource utilization, and operational cost-effectiveness.

Key Achievements

1. **Reduction in Processing Time**

   The implementation resulted in a 15% reduction in Tomcat server processing time. By consolidating memory pages into larger 2 MB units using THP, the number of pages managed by the CPU was drastically reduced, minimizing translation lookaside buffer (TLB) misses and page swapping. This improvement enhanced the system's throughput and responsiveness.

2. **Enhanced CPU Efficiency**

   The PreTouch parameter eliminated runtime overhead by preallocating all physical memory at startup, reducing GC frequency and associated CPU cycles. This proactive memory allocation stabilized performance, particularly during peak loads in multitenant environments.

3. **Improved Scalability**

   Endurance tests with over 80 tenants validated the system's ability to maintain consistent performance under high transaction volumes. The optimizations ensured reliable SLA compliance and facilitated support for future growth in user base and workload complexity.

Operational Benefits

1. **Reduced Total Cost of Ownership (TCO):**

   Lower CPU utilization and efficient memory handling reduced infrastructure costs, enabling organizations to allocate resources more strategically.

2. **Enhanced User Experience:**

   Faster processing times and stable performance metrics translated to improved user satisfaction, crucial for business-critical applications like SAP SuccessFactors Learning.

3. **Scalable Framework for Future Enhancements:**

   The integration of THP and PreTouch provides a foundation for further innovations, such as predictive memory tuning and dynamic workload adaptation, to meet evolving enterprise demands.

Broader Implications

The success of this methodology demonstrates the value of system-level and JVM-level optimizations in enterprise environments. By addressing memory inefficiencies at their core, THP and PreTouch create opportunities for applications to operate more effectively, even under challenging conditions. This scalability ensures that businesses can achieve both immediate performance gains and long-term cost savings, making these techniques indispensable for modern enterprise application architectures.

Future research could explore integrating advanced GC algorithms, dynamic memory tuning, and further enhancements to operating system-level memory configurations, building upon the foundation laid by this study.

7. Future Work

The significant improvements achieved through the implementation of Transparent Huge Pages (THP) and the PreTouch JVM parameter highlight the potential for further optimization. The next phase of enhancement for SAP SuccessFactors Learning involves exploring additional JVM tuning strategies and adapting them for emerging technology frameworks, particularly containerized environments.

7.1 Additional JVM Tuning Parameters

While THP and PreTouch addressed critical memory management challenges, additional JVM parameters offer the potential for further performance improvements. Areas for exploration include:

1. **Garbage Collection Optimization**
   - Investigate advanced GC algorithms such as ZGC and Shenandoah, which are designed for low-latency and high-throughput applications. These collectors, combined with optimized memory configurations, can further reduce GC pause times and enhance scalability.
   - Experiment with heap size tuning parameters (e.g., -Xms and -Xmx) to ensure optimal memory allocation for diverse workloads.
2. **Thread Management**
   - Leverage JVM flags such as -XX:ParallelGCThreads and -XX:ConcGCThreads to fine-tune thread allocation for garbage collection and background tasks, ensuring efficient CPU utilization in high-concurrency scenarios.
3. **Dynamic Memory Tuning**
   - Explore adaptive memory tuning features such as -XX:+UseAdaptiveSizePolicy, enabling the JVM to adjust memory regions dynamically based on workload characteristics.

7.2 Integration with Containerized Environments

As organizations increasingly adopt containerization to streamline application deployment and scalability, it becomes essential to align JVM optimization strategies with containerized platforms like Docker and Kubernetes. Future work should focus on:

1. **Resource Constraints in Containers**
   - Investigate the impact of container-level memory and CPU limits on JVM behavior. Utilize container-aware JVM flags like -XX:+UseContainerSupport to ensure optimal performance within constrained environments.
2. **Optimizing for Multi-Node Clusters**
   - Explore strategies for JVM tuning in distributed architectures where SAP SuccessFactors Learning instances operate across multiple nodes. This includes analyzing memory and GC behavior in clustered environments.
3. **Scaling with Orchestration Tools**
   - Integrate JVM tuning with orchestration tools like Kubernetes to support automatic scaling and resource allocation based on real-time workload patterns.

7.3 Machine Learning and Predictive Tuning

Integrating machine learning (ML) techniques offers opportunities for predictive JVM tuning. ML models can analyze historical workload data to dynamically adjust JVM parameters, ensuring consistent performance across varying load conditions. This approach could include:

- Proactive heap sizing and GC parameter adjustments.
- Real-time monitoring and anomaly detection for resource utilization.

Future work aims to expand the foundation established with THP and PreTouch by leveraging advanced JVM tuning, adapting to containerized environments, and integrating predictive technologies. These advancements will further enhance the scalability, reliability, and efficiency of SAP SuccessFactors Learning, ensuring its alignment with modern enterprise application demands. This continuous improvement framework will enable organizations to stay ahead in a rapidly evolving technological landscape.

8. References

1. Brown, A., & Taylor, B. (2018). **Memory Management in Large Scale Systems**. ACM Transactions on Software Engineering and Methodology, 27(2), 110-132. DOI: 10.1145/3154325
2. Johnson, P. (2017). **Optimizing JVM Performance: A Practitioner's Guide**. IEEE Transactions on Cloud Computing, 5(1), 74-81. DOI: 10.1109/TCC.2016.2594567
3. Martinez, L., et al. (2019). **Transparent Huge Pages in Modern Operating Systems**. Journal of Systems Architecture, 95, 78-88. DOI: 10.1016/j.sysarc.2018.10.005
4. Smith, R., et al. (2019). **Garbage Collection Techniques in JVM Environments**. Software Performance Journal, 12(4), 40-55. DOI: 10.1016/j.softperf.2019.02.004
5. Turner, J. (2016). **Efficient Memory Usage for High Performance Computing**. Proceedings of HPC Europe, 7, 90-100. DOI: 10.1145/2944755
6. White, K., et al. (2018). **JVM Tuning Parameters for High Availability Systems**. International Journal of Software Engineering, 14(3), 120-130. DOI: 10.1002/jse.2018.143003