Z Garbage Collector (ZGC) Implementation Recommendations

For Entré Software and Apache Tomcat

LT-3126 | 25455

Contents

Executive Summary

DMP recommends that customers running systems with 8GB or more of heap memory migrate from G1GC (Garbage-First Garbage Collector) or any previous garbage collection strategy to the Z Garbage Collector (ZGC). This recommendation is based on ZGC's superior performance characteristics, particularly its ability to maintain ultra-low pause times regardless of heap size, making it ideal for long-running enterprise applications such as Entré and Apache Tomcat.

ZGC represents a significant advancement in garbage collection technology, offering predictable sub-millisecond pause times even with multi-terabyte heaps. For production environments where application responsiveness is critical, ZGC provides substantial improvements over traditional garbage collectors without requiring extensive tuning or configuration.

What is the Z Garbage Collector (ZGC)?

The Z Garbage Collector (ZGC) is a scalable, low-latency garbage collector developed by Oracle and integrated into the Java platform. First introduced as an experimental feature in JDK 11 and made production-ready in JDK 15, ZGC is specifically designed to meet the demands of modern applications that require large heaps and minimal pause times.

Core Technology

ZGC employs several advanced techniques that distinguish it from traditional garbage collectors:

- **Concurrent Operations:** ZGC performs all expensive garbage collection work concurrently with application threads, ensuring that the application continues running without significant interruption.
- **Colored Pointers:** ZGC uses colored pointers (metadata bits stored directly in object references) to track object states, enabling efficient concurrent processing without requiring separate metadata structures.
- **Load Barriers:** Through load barriers, ZGC can relocate objects in memory while the application is running, allowing it to compact the heap without stopping application threads.
- **Region-Based Memory Management:** ZGC divides the heap into regions of varying sizes (small, medium, and large), optimizing memory allocation and garbage collection for different object sizes.

These technologies work together to achieve ZGC's primary goal: keeping garbage collection pause times below 10 milliseconds, regardless of heap size or the amount of live data.

Why Use ZGC?

Performance Benefits

ZGC delivers several critical performance advantages for enterprise applications:

- **Ultra-Low Pause Times:** ZGC consistently maintains pause times under 10 milliseconds, typically achieving sub-millisecond pauses. This ensures predictable application response times regardless of garbage collection activity.
- **Scalability:** Unlike G1GC, whose pause times increase with heap size, ZGC's pause times remain consistent whether managing an 8GB heap or a 4TB heap. This scalability is crucial for applications that may need to grow over time.

- **Improved Application Throughput:** By performing most garbage collection work concurrently, ZGC minimizes the impact on application throughput, allowing Entré and Tomcat to maintain consistent performance under load.
- **Reduced Latency Variance:** Applications experience more predictable performance characteristics, with fewer unexpected pauses that can impact user experience or service-level agreements.
- **Memory Efficiency:** ZGC can return unused memory to the operating system, making it more efficient in environments where memory is a shared resource.

Specific Advantages for Entré and Apache Tomcat

For long-running server applications like Entré software and Apache Tomcat, ZGC offers advantages:

- **Consistent Response Times:** Web applications and services require predictable response times. ZGC eliminates the long garbage collection pauses that can cause request timeouts or poor user experience.
- **Better Resource Utilization:** The concurrent nature of ZGC means that CPU resources are used more efficiently, with less time spent in stop-the-world phases.
- **Simplified Tuning:** ZGC requires minimal tuning compared to G1GC. There are no complex parameters for pause time goals, region sizes, or occupancy thresholds to configure.
- **Future-Proof Architecture:** As application memory requirements grow, ZGC's architecture ensures that performance remains consistent without requiring reconfiguration or retuning.

System Requirements and Prerequisites

Operating System Requirements

ZGC requires a modern operating system with support for specific memory management features. For Windows environments, the following minimum versions are required:

- Windows Server 2019 or newer
- Windows 10 Build 1803 (April 2018 Update) or newer

These version requirements are necessary because ZGC relies on operating system features for memory management that are not available in earlier Windows versions.

Verifying System Compatibility

Before implementing ZGC in production, verify that your operating system supports ZGC by running the following test from the command line:

```
java -XX:+UseZGC -Xms24G -Xmx24G -version
```

If the command completes without errors and displays the Java version information, your operating system can run ZGC. If you receive an error message, your operating system does not meet the requirements and must be upgraded before ZGC can be used.

Hardware Recommendations

DMP recommends ZGC for systems with 8GB or more of heap memory. While ZGC can technically work with smaller heaps, its advantages become most apparent with larger memory allocations. For optimal performance, ensure that:

- The system has sufficient physical RAM to support the configured heap size plus operating system and other application requirements
- Multiple CPU cores are available, as ZGC uses concurrent threads for garbage collection operations
- The Java Development Kit (JDK) is version 15 or later for production use (JDK 11-14 can use ZGC experimentally)

How to Configure ZGC

Basic Configuration

Configuring ZGC is straightforward and requires minimal parameters compared to other garbage collectors. The basic configuration consists of enabling ZGC and setting the heap size parameters.

Single-Line Configuration

You can configure ZGC and memory constraints in a single line:

```
-XX:+UseZGC -Xms24G -Xmx24G
```

Multi-Line Configuration

Alternatively, you can configure these parameters on separate lines for better readability in configuration files:

```
-XX:+UseZGC
-Xms24G
-Xmx24G
```

Memory Configuration Best Practices

Setting Xms and Xmx to Equal Values

DMP strongly recommends setting your initial heap size (Xms) and maximum heap size (Xmx) to the same value. This practice provides several important benefits:

- **Eliminates Heap Resizing Overhead:** When Xms and Xmx are equal, the Java Virtual Machine allocates the full heap at startup and never needs to resize it during runtime. Heap resizing operations can cause performance degradation and temporary pauses, which are completely avoided with this configuration.
- **Improves Startup Performance:** The full heap is allocated immediately at application startup, reducing warm-up time. This is particularly beneficial for long-running services like Entré and Apache Tomcat, where startup performance is important, but the application will run for extended periods.
- **Predictable Memory Usage:** System administrators can accurately plan memory allocation knowing exactly how much memory the application will consume, rather than dealing with a variable heap size.
- **Optimal for ZGC Design:** ZGC is specifically designed for large heaps and works best when given a consistent memory footprint. Setting Xms equal to Xmx aligns with ZGC's design philosophy.

For example, if you determine that your application requires 24GB of heap memory, configure both parameters identically: `-Xms24G -Xmx24G`

Removing Obsolete Xmn Configuration

If your current configuration includes the `-Xmn` parameter (which sets the young generation size), **this parameter should be removed when migrating to ZGC**. ZGC does not use generational garbage collection and manages heap regions dynamically. The Xmn parameter is specific to generational collectors like G1GC and will be ignored or may cause unexpected behavior with ZGC.

Removing G1GC-Specific Configuration Flags

When migrating from G1GC to ZGC, it is essential to remove all G1GC-specific tuning flags from your configuration. These flags are incompatible with ZGC and will either be ignored or cause configuration errors. The following sections detail the G1GC flags that must be removed.

Primary G1GC Tuning Flags to Remove

-XX:MaxGCPauseMillis=...

This flag sets a target for maximum garbage collection pause time in G1GC. ZGC does not use this parameter because it targets low pause times by design, automatically maintaining pause times below 10 milliseconds without requiring explicit configuration.

-XX:G1HeapRegionSize=...

This flag controls the size of heap regions in G1GC. ZGC does not use the same region-based model and manages its own internal heap organization dynamically.

-XX:InitiatingHeapOccupancyPercent=...

This flag determines when G1GC should start a concurrent marking cycle based on heap occupancy. ZGC uses a completely different mechanism for triggering garbage collection cycles and does not rely on occupancy thresholds.

-XX:ParallelGCThreads=...

This flag sets the number of threads used for parallel garbage collection phases. ZGC manages its own threading model internally and automatically determines the optimal number of threads based on available CPU cores.

-XX:ConcGCThreads=...

This flag sets the number of threads used for concurrent garbage collection work in G1GC. This parameter is not applicable to ZGC, which manages concurrent operations using its own internal mechanisms.

-XX:+ExplicitGCInvokesConcurrent

This flag causes G1GC to perform concurrent garbage collection when System.gc() is called explicitly. ZGC handles System.gc() calls differently and this flag is not needed.

Additional G1GC-Related Flags

-XX:+UnlockExperimentalVMOptions

This flag was required to enable ZGC in JDK versions 11-14 when ZGC was still experimental. In JDK 15 and later, ZGC is production-ready, and this flag is no longer required. If you are using JDK 15 or later, this flag can be safely removed (At the time of this writing, you should be running JDK 17 for Entré). If you are using an earlier version, the flag may still be present but will not interfere with operation.

-XX:+UseStringDeduplication

This flag enables string deduplication, a feature specific to G1GC that identifies and removes duplicate String objects to save memory. This feature only works with G1GC and must be removed when using ZGC.

ZGC-Compatible JVM Flags

While ZGC requires minimal configuration, several optional flags are available for specific scenarios.

Required Flags

-XX:+UseZGC

Enables the Z Garbage Collector. This is the only required flag to use ZGC.

-Xms<size>

Sets the initial heap size. Values can range from 512M for small applications to 32G or more for large enterprise systems. **DMP recommends matching -Xmx for stability**.

Example: `–Xms24G`

-Xmx<size>

Sets the maximum heap size. Values should be based on application workload requirements and available system memory. For most Entré and Tomcat deployments, values between 8G and 32G are common.

Example: `–Xmx24G`

Optional Advanced Flags

The following flags provide additional control over ZGC behavior. However, **DMP does not recommend using these flags for typical Entré and Apache Tomcat deployments** as the default ZGC configuration is optimized for most workloads.

-XX:SoftMaxHeapSize=<size>

This flag suggests a soft upper limit for heap usage. ZGC may release memory above this threshold back to the operating system. Values typically range from 50% to 90% of -Xmx.

DMP recommendation: This flag is unnecessary for non-containerized environments where Entré and Tomcat are running on dedicated servers. The benefit of returning memory to the operating system is minimal when the application is the primary memory consumer.

-XX:+ZUncommit

Enables ZGC to uncommit unused memory and return it to the operating system.

DMP recommendation: This flag is not recommended for typical DMP deployments. Since customers do not use containerization and typically run dedicated servers for Entré and Tomcat, there is no benefit to returning memory to the operating system. Additionally, uncommitting memory adds minor overhead that could be avoided. The default behavior of maintaining the allocated heap is more appropriate for these environments.

-XX:+UnlockExperimentalVMOptions

Unlocks experimental JVM options.

DMP recommendation: This flag is only necessary when using JDK versions earlier than 15. Since ZGC became production-ready in JDK 15, this flag should not be used with JDK 15 or later. If you are using JDK 11-14, this flag must be included, but DMP recommends upgrading to JDK 17 or later for production deployments.

DMP Official Recommendation on Additional ZGC Switches

After thorough analysis of the optional ZGC configuration flags, DMP's official position is that customers should NOT use the additional switches (SoftMaxHeapSize, ZUncommit, UnlockExperimentalVMOptions) in typical deployments.

Rationale

This recommendation is based on the following considerations specific to DMP customer environments:

- **No Containerization:** Since DMP customers do not use containerized environments, the memory management features that these flags provide are not beneficial. Features like uncommitting memory to return it to the operating system are designed for containerized environments where multiple applications compete for shared memory resources.
- **Dedicated Server Architecture:** Entré and Tomcat typically run on dedicated servers where they are the primary memory consumer. Returning unused memory to the operating system provides no advantage in this scenario and only adds unnecessary overhead.
- **Optimal Default Configuration:** ZGC's default configuration is carefully tuned for most workloads. The addition of optional flags introduces complexity without providing meaningful performance improvements for the majority of DMP deployments.
- **Simplified Operations:** Keeping the configuration simple reduces the likelihood of misconfigurations and makes troubleshooting easier if issues arise.
- **Production Stability:** With JDK 17 or later, ZGC is production-ready without experimental flags. Using only the essential configuration parameters ensures maximum stability.

Recommended Minimal Configuration

For the vast majority of DMP customers, the recommended ZGC configuration consists of only three parameters:

```
-XX:+UseZGC -Xms100G -Xmx100G
```

Replace the heap size values (100G in this example) with the appropriate size for your application's memory requirements, ensuring that Xms and Xmx are set to the same value.

Implementation Guidelines

Migration Process

Migrating from G1GC or another garbage collector to ZGC should be performed systematically to ensure a smooth transition. DMP recommends the following process:

- **Verify System Compatibility:** Run the compatibility test command to confirm your operating system supports ZGC before proceeding with any configuration changes.
- **Document Current Configuration:** Record your current JVM parameters and performance metrics to enable comparison after migration and provide a rollback reference if needed.
- **Test in Non-Production Environment:** Deploy the new ZGC configuration in a development or staging environment first. Monitor the application for several days under typical workload conditions to identify any issues before production deployment.
- **Update Configuration Files:** Modify your JVM startup parameters by adding -XX:+UseZGC, removing all G1GC-specific flags, and setting Xms and Xmx to equal values appropriate for your application.
- **Schedule Planned Downtime:** Coordinate a maintenance window for production deployment, as the application must be restarted to apply the new garbage collector configuration.
- **Monitor Performance:** After deployment, closely monitor application performance, response times, and garbage collection metrics for at least several days. Pay particular attention during peak usage periods.
- **Maintain Rollback Capability:** Keep your previous configuration readily available for at least two weeks after production deployment to enable quick rollback if unexpected issues arise.

Configuration Examples

The following examples demonstrate proper ZGC configuration for different memory requirements:

Small Deployment (8GB heap):

```
-XX:+UseZGC -Xms8G -Xmx8G
```

Medium Deployment (16GB heap):

```
-XX:+UseZGC -Xms16G -Xmx16G
```

Large Deployment (32GB heap):

```
-XX:+UseZGC -Xms32G -Xmx32G
```

Enterprise Deployment (64GB heap):

```
-XX:+UseZGC -Xms64G -Xmx64G
```

Monitoring and Troubleshooting

Monitoring ZGC Performance

After implementing ZGC, monitor the following metrics to verify proper operation and optimal performance:

- **Garbage Collection Pause Times:** ZGC pause times should consistently remain below 10 milliseconds. If you observe pause times exceeding this threshold, contact DMP support for assistance.
- **Application Response Times:** Monitor application response times during and after garbage collection events. Response times should be more consistent compared to G1GC, with fewer latency spikes.
- **Heap Usage:** Observe heap usage patterns to ensure the allocated heap size is appropriate for your workload. The heap should not regularly approach full capacity.
- **CPU Utilization:** ZGC performs concurrent work, so some CPU utilization for garbage collection is normal. However, if you observe unexpectedly high CPU usage, this may indicate a need for heap size adjustment.

Enabling GC Logging

For detailed garbage collection analysis, enable ZGC logging by adding the following parameters:

```
-Xlog:gc*:file=gc.log:time,uptime:filecount=5,filesize=100M
```

This configuration creates rotating log files that capture detailed garbage collection information for troubleshooting and performance analysis.

Common Issues and Solutions

OutOfMemoryError Exceptions

If the application throws OutOfMemoryError exceptions after migrating to ZGC, the heap size may be insufficient. Increase both Xms and Xmx by 25-50% and monitor.

Application Performance Degradation

If application performance is worse with ZGC than with G1GC, verify that all G1GC-specific flags have been removed from the configuration. Conflicting flags or incorrect heap sizing can impact performance. Review the configuration against the examples in this document.

High CPU Usage

Elevated CPU usage after implementing ZGC may indicate that the heap size is too small, causing ZGC to work harder to maintain free memory. Consider increasing the heap size. Alternatively, high CPU usage during initial operation may be normal as ZGC performs initial compaction work and should stabilize over time.

Conclusion

The Z Garbage Collector represents a significant advancement in Java garbage collection technology, offering predictable low-latency performance that scales effectively to large heaps. For DMP customers running Entré software and Apache Tomcat with 8GB or more of heap memory, migrating to ZGC provides substantial benefits in application responsiveness and predictability.

The simplicity of ZGC configuration, combined with its superior performance characteristics, makes it an ideal choice for production environments. By following the guidelines in this document (particularly the recommendation to use only the essential configuration flags and to set Xms and Xmx to equal values) customers can achieve optimal performance with minimal complexity.

DMP strongly encourages all eligible customers to plan and implement the migration to ZGC. The performance improvements, operational simplicity, and future-proof architecture make ZGC the recommended garbage collector for modern Java applications. For customers who require assistance with the migration process or have questions about their specific configuration, DMP support is available to provide guidance and ensure a successful transition.

Support and Additional Resources

For assistance with ZGC implementation or to discuss your specific deployment requirements, please contact DMP Technical Support or DMP Software Services. Our teams can provide personalized guidance based on your environment and workload characteristics to ensure optimal configuration and performance.

Additional Resources:

- **Oracle ZGC Documentation:** Official technical documentation and updates from Oracle
- **DMP Technical Support:** Direct assistance from DMP technical support specialists who are familiar with your deployment.
  Email: techsupport@dmp.com Phone: 888-436-7832
- **DMP Software Services:** Direct assistance from DMP engineers who are familiar with your deployment. Email: softwareservices@dmp.com