Comparison of Garbage Collection Algorithms in Java

1. Serial Garbage Collector (Single-Threaded)

Description:

A simple, stop-the-world, single-threaded garbage collector designed for single-threaded applications or small heaps.

Key Features:

- Works in a single thread for both minor and major collections
- Suitable for environments with low memory and single-threaded applications

Pros:

- Simple and efficient for small applications
- Low overhead

Cons:

- Pauses the entire application during collection (stop-the-world)
- Not scalable for modern multi-threaded applications

Best Use Case:

Desktop applications or development environments with small heap sizes

---

2. Parallel Garbage Collector (Throughput Collector)

Description:

A multi-threaded collector that focuses on maximizing throughput by performing garbage collection in parallel.

Key Features:

- Multiple threads for both minor and major collections
- Balances application performance and GC overhead

Pros:

- Efficient for applications with high throughput requirements

- Can handle larger heaps

Cons:

- Stop-the-world pauses still occur
- Less predictable GC behavior compared to CMS or G1

Best Use Case:

Batch processing or compute-heavy server applications

---

3. Concurrent Mark-Sweep (CMS) Garbage Collector

Description:

Reduces pause times by performing concurrent phases for garbage collection.

Key Features:

- Concurrent marking and sweeping with minimal pauses
- Aimed at low-latency applications

Pros:

- Reduced pause times compared to Serial and Parallel collectors
- Handles concurrent processing

Cons:

- Higher CPU usage due to concurrent threads
- Possible fragmentation issues

Best Use Case:

Low-latency applications such as web servers

---

4. Garbage First (G1) Garbage Collector

Description:

A region-based collector designed to offer predictable and low-latency pause times.

Key Features:

- Divides the heap into regions and prioritizes GC based on predicted gain
- Concurrent collection phases

Pros:

- Better pause time control
- Balances throughput and latency
- Handles larger heaps efficiently

Cons:

- Complexity in tuning
- Overhead in managing regions

Best Use Case:

Applications requiring predictable low latency and large heap sizes

---

5. Z Garbage Collector (ZGC)

Description:

Ultra-low latency collector designed for massive heaps with near-real-time application response.

Key Features:

- Pauses limited to a few milliseconds
- Supports heaps ranging from small sizes to multi-terabyte heaps
- Mostly concurrent

Pros:

- Minimal pause times (~10ms or less)
- Scales well for massive heaps
- Handles dynamic memory allocation efficiently

Cons:

- Higher CPU usage compared to simpler collectors
- Available only in newer versions of Java (starting from JDK 11)

Best Use Case:

Large-scale, real-time applications like financial systems or AI/ML platforms

---

Comparison Table

Collector	**Pause Time**	**Concurrency**	**Throughput**	**Heap Size**	**Best For**
Serial	High	Single-threaded	Low	Small	Simple apps
Parallel	Moderate	Multi-threaded	High	Medium-Large	Batch tasks
CMS	Low	Concurrent	Moderate	Medium	Web servers
G1	Low	Concurrent	High	Large	Low latency
ZGC	Very Low	Highly Concurrent	Moderate	Massive	Real-time

Conclusion

- Choose **Serial GC** for simple, single-threaded apps.
- Prefer **Parallel GC** for compute-heavy batch operations.
- Use **CMS GC** for low-latency web servers.
- Select **G1 GC** for large-scale apps with predictable pause requirements.
- Opt for **ZGC** for real-time, large-heap applications needing ultra-low latency.