www.ijecs.in
International Journal Of Engineering And Computer Science
Volume 13 Issue 02 February 2024, Page No. 26015-26020
ISSN: 2319-7242 DOI: 10.18535/ijecs/v13i02.5087

Memory management in Hazelcast

Filisov Denis Alexandrovich

Senior Software Engineer. https://career.luxoft.com/
Belgrade, Serbia

Abstract
Hazelcast, an in-memory storage system, relies on efficient memory management to ensure seamless scalability and high-performance computing for distributed applications. Memory management in Hazelcast is a critical aspect that ensures optimal resource utilization, minimizes latency, and increases overall system reliability. This complex process involves allocating, releasing, and organizing memory resources according to the notes of the Hazelcast cluster. Understanding memory management in Hazelcast is important for developers and architects seeking to harness the full potential of this advanced in-memory data grid for their distributed applications.
The purpose of the work is to consider Memory management in Hazelcast. To achieve this goal, the scientific works of domestic and foreign authors were used.

Keywords: Memory management, Hazelcast, programming, IT, software, modern technologies, digitalization.

Introduction

In the realm of distributed computing, one of the most pressing challenges confronting today's enterprises involves efficiently managing and processing vast volumes of data in real-time. The surge in data-intensive applications across various domains necessitates a solution that not only handles large-scale data but also delivers high performance with low latency. Hazelcast IMDG (In-Memory Data Grid), an open-source leader in in-memory operations, emerges as a pivotal response to these challenges. Engineered in Java, Hazelcast IMDG excels in storing extensive data volumes across a flexible, distributed cluster, boasting a rich array of features for effective data interaction.

The core philosophy of Hazelcast centers on its exclusive operation within memory, ensuring low and predictable latency along with consistent throughput. This architecture is crucial for applications demanding rapid data access, such as caching layers, web session storage, and efficient fraud detection mechanisms. However, as a Java-based solution, Hazelcast faces the inherent unpredictability of Java's Garbage Collection (GC) algorithms, which can impose constraints on heap size and lead to GC pause-induced performance hiccups. Addressing this limitation, Hazelcast has evolved its enterprise offerings with innovations like High-Density Memory Store (HDMS or HD), a memory storage solution outside of Java's purview, circumventing GC-related issues. This introduction of HDMS

represents a significant stride in memory management, enabling larger data sizes without the traditional GC drawbacks.

Furthermore, the integration of cutting-edge technologies like Intel Optane Persistent Memory, first released in 2019, offers a new dimension in memory resource management. These modules, known for their non-volatility and higher capacity compared to traditional DRAM modules, provide a cost-effective and performance-oriented solution, making them highly attractive for Hazelcast's distributed computing scenarios [1].

Methodology

This paper delves into the intricate challenges and solutions in managing large-scale data with Hazelcast, exploring its advanced memory management techniques, the integration of diverse memory resources like DRAM and PMem, and the performance implications therein. By highlighting these issues and Hazelcast's innovative approaches, this study aims to offer valuable insights and solutions for businesses grappling with the complexities of large-scale, in-memory data management in a rapidly evolving digital landscape.

1. Working with big data in Java

Given that Hazelcast is written in the Java programming language, it detects sensitivity to the unpredictability of the garbage collection algorithm (GC) that is used by the Java virtual machine. It is widely known that standard PS algorithms impose restrictions on the useful volume of the heap, which can manifest itself in pauses of the PDS with a stop of the world.

In order to overcome the limitations associated with GC and ensure stable and predictable response times, Hazelcast upgraded its enterprise offering several years ago by introducing in-memory storage that stores data outside Java. This feature, known as high-density data storage (HDPE or simply HD), is also known as stand-alone storage. Despite the involvement of GC, the data stored by Hazelcast in its HD storage avoids the impact of GC algorithms, which allows them to be large without encountering the aforementioned GC problems.

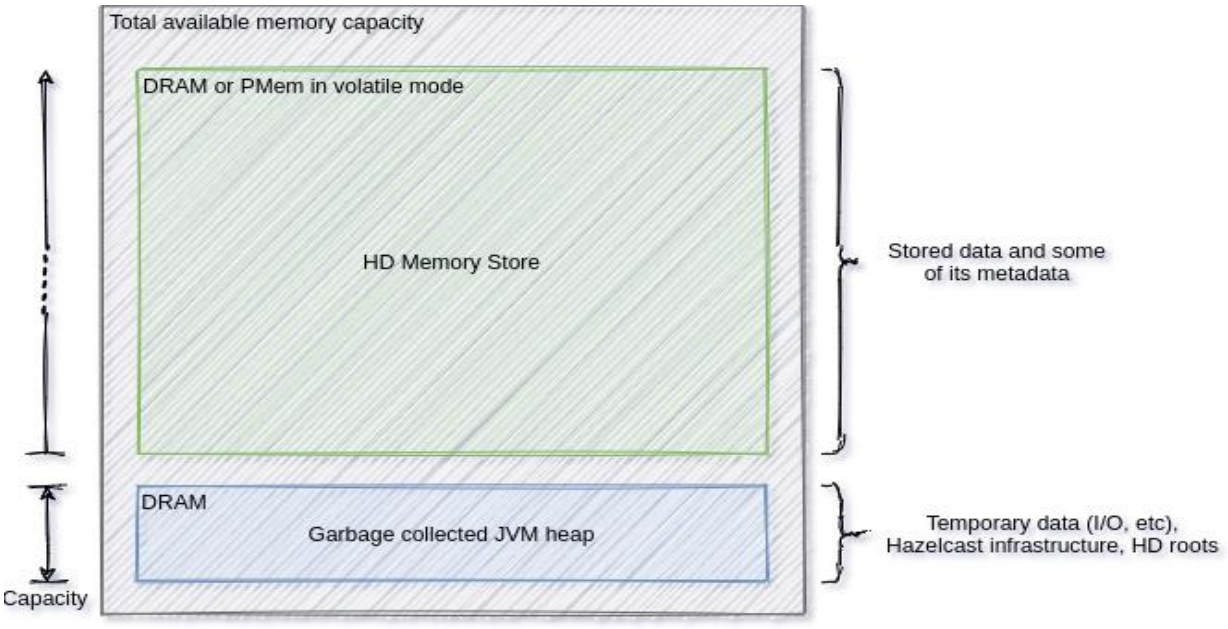

Fig.1. Memory segmentation using HD

Intel Optane Persistent Memory, released for the first time in 2019, offers non-volatile memory modules compatible with DIMMs. Despite the fact that the main achievement of these modules is their stable functionality, using them in the App Direct mode, which does not depend on power supply, is also attractive in some scenarios.

These modules are provided with higher capacity compared to DRAM DIMM modules, which increases the total available capacity. At the same time, the modules provide performance close to DRAM, at a lower cost per gigabyte. All these factors make Intel Optane PMem modules attractive for Hazelcast [2].

For managing heterogeneous memory resources such as DRAM and PMem, Memkind provides an excellent solution. Internal tracking of the type of allocated memory helps to avoid the difficulties of managing different memory allocators and reduces fragmentation.

The collaboration between the PMDK team and Hazelcast has led to improved integration of PMem into high-density memory storage. The use of Memkind and optimizations in the PMDK library have improved performance and reduced Hazelcast downtime. This partnership has also pushed for improvements to the Memkind library, making it more autonomous and providing new verification methods that simplify the use of PMem.

2. Unified PMEM system

As mentioned earlier, Hazelcast actively uses various types of PMEM, including MEMKIND_DAX_KMEM_ALL. MEMKIND_DAX_KMEM_ALL is an easy-to-use view that defines a single system even in multi-socket systems. On the other hand, the DAX mode of the file system, on which the PMEM view is based, requires special attention.

A unified system in FS-DAX mode, where there are many permanent memory devices, even if they alternate, one device per socket. Figure 2 shows the file system-based solution used by memkind to support PMem devices. Each PMem device is presented to the operating system as a block device based on a DAX-enabled file system. A block device can be mounted in various locations, so the user must create a PMEM memory type for a specific path. In this mode, in order to use all the PMem memory available in the system, the application must create separate heaps for each socket (i.e., each PMem device). In addition to individually managing them, the application is also responsible for ensuring the availability of NUMA-aware distributions.

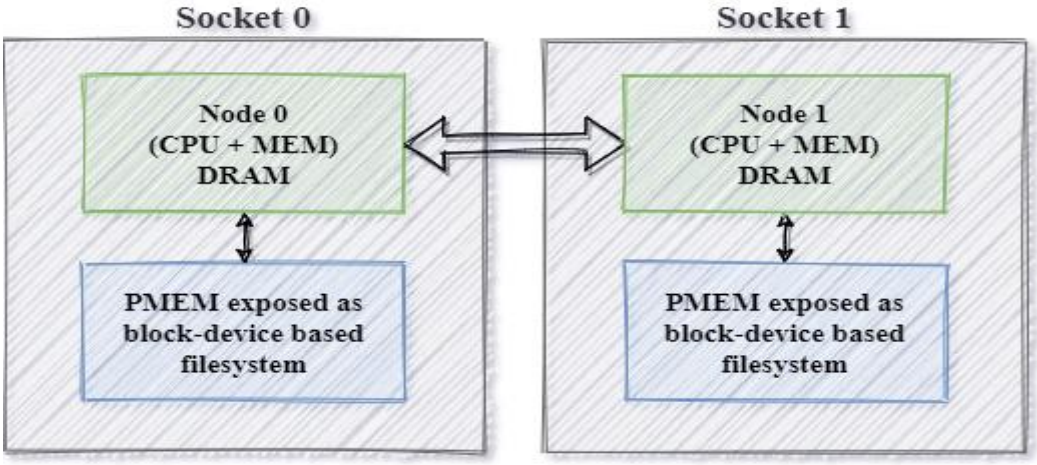

Fig.2. PMEM support in DAX file system mode

This strategy is the optimal choice for many use cases and is used as the default strategy. However, in some scenarios, for example, when working with big data, performing calculations for multiple records, or remote access, which entails significant performance costs, it is worth considering using a strategy that supports NUMA. It binds each Hazelcast operational thread to a single NUMA node and performs each allocation as a PMEM, local to the allocator thread. In such cases, it helps to increase the response time by reducing the delay of LLC misses. To set up a distribution strategy, refer to the Hazelcast reference guide.

These strategies are focused on the performance of accessing permanent memory, while maintaining the importance of being able to use all available capacity.

3. Performance

The performance of PMem modules is between the performance of DRAM modules and storage devices. The bandwidth of Intel Optane's permanent memory is lower than that of DRAM modules, while the access time is longer. Some might suggest that PMem is inefficient compared to DRAM in terms of performance. However, Hazelcast tests show the opposite. Despite a slight decrease in the performance of PMem devices, in many use cases Hazelcast recorded PMem performance results close to DRAM results. There are several reasons for this [3].

Since persistent memory accesses are cached in processor cache layers in the same way as DRAM accesses, cache localization techniques used to speed up DRAM accesses are also effective for PMem. Hazelcast uses these methods to switch between DRAM and PMem accesses to cache accesses where possible. This makes it immaterial whether the cache lines are used by DRAM or Intel Optane permanent memory.

Another factor is that Hazelcast is a distributed system that interacts with a variety of network communications, including serialization, copying to memory, and internal exchanges between threads. Errors added by the JVM, such as GC pauses, and the operating system also affect performance. All of this creates costs that can potentially exceed the additional costs of accessing PMem compared to accessing DRAM. In other words, Intel Optane's permanent memory characteristics can help achieve similar performance compared to DRAM when working in a distributed environment. The key conclusion of Hazelcast is that NUMA locality plays an important role. Figure 4 clearly demonstrates the potential benefits of accessing Intel Optane's NUMA-local permanent memory in FS-DAX mode. The diagram compares the bandwidth of a Hazelcast node with the same caching load, but with different settings. The benchmark includes two operations - get and put operations in Hazelcast IMap (Hazelcast key-value data structure), both are performed with a 50% probability to read and update values. The legend encodes these settings as follows:

DRAM/OPTANE/OPTANE-NUMA: Whether Hazelcast data is stored in DRAM or Optane. OPTANE stands for the previously described cyclic allocation strategy, while OPTANE-NUMA stands for a NUMA-based allocation strategy.

10K: The size of the records used in the test.

100G: The total size of the data that the Hazelcast member is working with.

1: The iteration number of the test with the specified settings. In this case, there is always one.

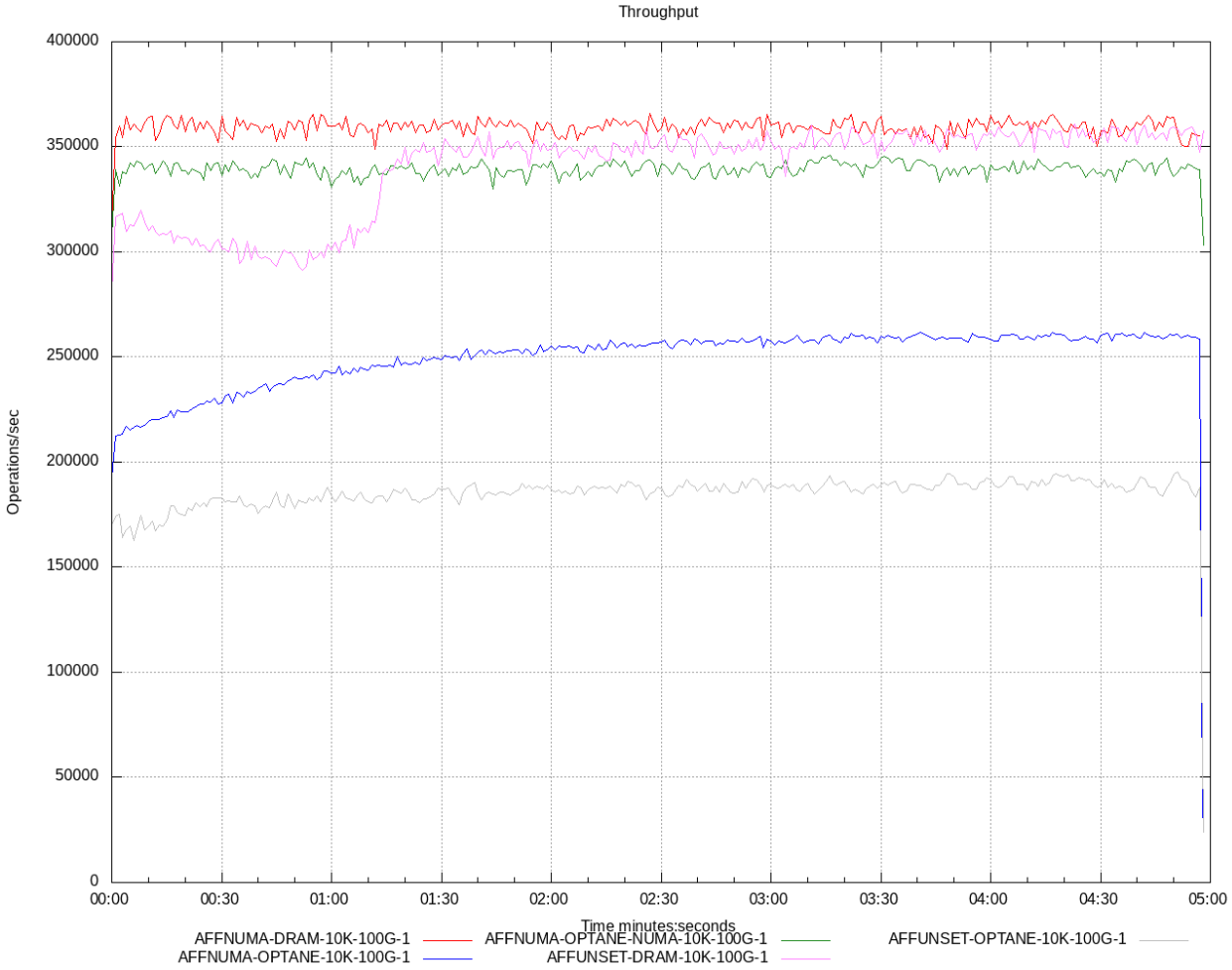

Fig.3. Bandwidth of a single Hazelcast instance with different settings

The diagram shows a dramatic change in Hazelcast bandwidth just by changing the NUMA-local access to Intel Optane permanent memory from 50% (blue line) to 100% (green line). It also shows that there is not such a big difference in the case of DRAM (pink and red lines). In addition to being less sensitive to NUMA localization of DRAM, another reason is that the operating system may decide that it is best to store data and the code using it on the same NUMA node. This is also a good attribute of KMEM-DAX mode, pages supported by PMem used in KMEM-DAX mode are subject to the usual Linux kernel memory allocation policies. This is not the case if FS-DAX mode is used, because in this mode, the application's access pattern to PMem is completely opaque to the operating system, since the management of the heap supported by PMem is completely performed in user space [4].

4. Data Integrity

Hazelcast IMDG is coming forward again, solving data integrity issues in the face of network failures often associated with problems of the CAP theorem. Hazelcast now uniquely provides both a consistency subsystem (CP) for concurrency structures that prefer consistency, and an availability subsystem (AP) for data storage structures, giving preference to accessibility. By providing AP and CP subsystems, Hazelcast gives customers the ability to fine-tune the in-memory data network to meet application requirements, while providing a consistent and stable user experience. The CP subsystem has successfully passed the standard Jepsen data correctness tests.

"This breakthrough allows Hazelcast to achieve the highest standards of data correctness in the industry," said Hazelcast CTO Greg Luck. "Competitors in this field face serious difficulties. Solving these problems will allow customers to use Hazelcast's parallel data structures to build their applications, minimizing dependence on additional clustering software and simplifying operations."

This new capability for the AP and CP subsystems is a technological advance and builds on the successes achieved in IMDG 3.10, which solve the problems of "brain separation" - a condition that poses a threat to data integrity in a distributed system.

Hazelcast IMDG 3.12 now supports first-class JSON data structures in all language clients, which is often used in document databases. Internal tests of mixed workloads confirm that Hazelcast provides 4 times more bandwidth compared to popular NoSQL solutions. This makes Hazelcast a multi-model platform, complementing the slower NoSQL offerings for low-latency scenarios of accessing short-lived information such as web sessions or transactions.

Hazelcast IMDG Enterprise also introduces "blue-green" deployments for gradual software updates and provides automatic disaster recovery with fault tolerance. In case of a failure on the main site, Hazelcast will automatically switch to the additional DR site, which ensures security for the most demanding organizations [5].

Conclusion

Thus, memory management in Hazelcast represents an important aspect of its functionality, providing efficient and scalable distributed data storage. The memory management mechanisms in Hazelcast provide users with the flexibility to optimize resource usage and ensure stable performance under high load conditions.

Hazelcast, as a tool for building distributed systems, provides powerful memory management tools while providing transparency to developers. With its help, developers can create high-performance applications that can adapt to the growing needs of society.

References

1. Optimized Memory Management for a Java-Based Distributed In-Memory System. [Electronic resource] Access mode: https://coconucos.cs.uni-duesseldorf.de/forschung/pubs/2019/SCRAMBL19.pdf .– (accessed 01/25/2024).
2. USING MEMKIND IN HAZELCAST. [Electronic resource] Access mode: https://pmem.io/blog/2021/02/using-memkind-in-hazelcast /.– (accessed 25.01.2024).
3. 3 Recommendations for using Hazelcast memory. [Electronic resource] Access mode: https://docs.cloudera.com/cem/1.6.0/efm-infrastructure-and-performance/topics/cem-hazelcast-memory-considerations.html. – (accessed 01/25/2024).
4. Hazelcast In-Memory Breakthroughs Redefine Data Integrity for Critical Workloads. [Electronic resource] Access mode: https://www.enterpriseai.news/2019/02/27/hazelcast-in-memory-breakthroughs-redefine-data-integrity-for-critical-workloads/. – (accessed 25.01.2024).
5. Hazelcast: Java distributed memory grid framework (platform). [Electronic resource] Access mode: https://programmerall.com/article/56121008655 /.– (accessed 25.01.2024).