Software Development Tech Bites

In-memory Caching using Redis

By Mustafa Supuk December 22, 2023 No Comments

The importance of computer memory utilization

The CPU and memory are the main components of any computer system. Computer memory stores data and program instructions, temporarily or permanently, that the CPU processes. In CPU-intensive applications with large amounts of data being processed, memory usually becomes the bottleneck, resulting in a significant overall performance decrease. Memory is organized in a hierarchical structure (Figure 1). For the data to be available

performance degradation. To achieve this, it is necessary to pay attention to memory utilization.

The memory hierarchy

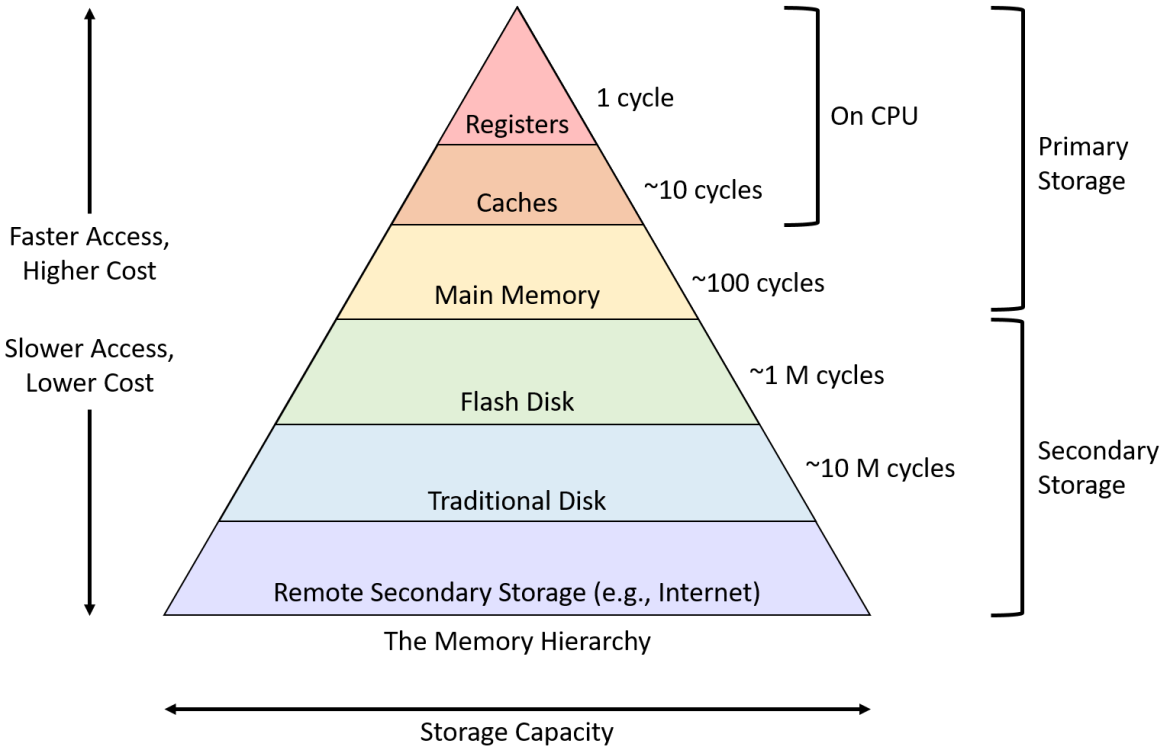

Figure 1. The memory hierarchy (source)

The memory hierarchy (Figure 1) shows how storage capacity, access time, and cost relate to each other between different memory layers. Faster access comes with a higher cost and a lower storage capacity. This structure proved to be the best option to make computer systems feasible at all. Obviously, all data from one layer cannot fit in the layer above, so we must ensure that only the most valuable data needed by our applications is available in data stores with faster access.

temporary, high-speed, transient storage system called cache. A cache is a software or hardware component aimed at storing data so that future requests for the same data can be served faster. Caching can help improve the performance, scalability, and cost-effectiveness of applications. Caches enable us to implement a mechanism to efficiently reuse previously retrieved or computed data. When we build a cache, we use a small amount of relatively fast storage at one level of the memory hierarchy to speed up access to a large and relatively slow storage at the next higher level (farther from the CPU) of the memory hierarchy. Reading the required data from caches is assumed to be faster than recomputing the result or reading it from the original, slower data store.

A cache hit occurs when the fetched data is present in the cache. Otherwise, a cache miss occurs. In the latter case, data still needs to be fetched from the slower storage, with the total access time being slower than directly accessing the slower storage. Therefore, a successful cache results in a high hit rate, meaning the data was present in the cache when fetched.

As a general computing concept, caching is implemented by different software and hardware components in multiple memory hierarchy layers. For example, all CPUs and GPUs, operating systems, and many applications like web browsers and databases use caches.

In-memory caching is a technique where frequently accessed data is stored in main memory instead of retrieved from disk or remote storage. Data can be cached in memory by caching systems like Redis.

About Redis

Redis, which stands for Remote Dictionary Server, is an open-source, in-memory data structure store used as a database, cache, message broker,

source version, more features like indexing, querying, and JSON data support are offered by Redis Stack. In addition, the Redis Enterprise version offers many enterprise-grade features like high availability, linear scaling, and geo-replicated distribution.

Data structures in Redis

Redis is organized as a key/value data store. Redis keys are unique strings used to identify and access value objects. Value objects can be of different types supported by multiple Redis data structures. While a key name can be very large (up to 512MB in length), extremely long keys should be avoided for performance reasons. A good practice for naming keys is to create and stick to a particular schema. For example, **type**:id is a common schema for identifying objects. Using this schema, the key for an object representing a user with an ID=7 would be **user**:7, while a token object that belongs to this user would have the key **user**:7:token.

Redis data can be managed using the Redis CLI tool or various Redis protocol implementations in many programming languages (complete list available at link). Using Redis data structures is simple and very similar to using standard, built-in data structures in programming languages. Basic operations supported by all data structures, like getting/setting values and inserting values in collections, are used most, while some data structures offer more possibilities. Redis Stack also offers advanced query functionalities using a simple syntax for complex queries, like numeric filters, tag filters, geo filters, polygon search, vector similarity search, fuzzy matching, etc.

Most used data types in Redis core:

also commands for incrementing/decrementing numerical values.

- Lists are linked lists of string values sorted by insertion order. They are frequently used to implement stacks and queues. Supported commands include adding elements to the head/tail of the list, removing elements from the head/tail, inserting elements, etc.
- Sets and sorted sets are collections of unique strings, with the latter maintaining order. In addition to standard commands for adding, removing, and getting the set size, set operations like union, intersection, and difference are also supported.
- Hashes are collections of key-value pairs, also known as HashMaps and dictionaries. Most commands that use a single key-value pair have constant time complexity.
- Streams are data structures that act like an append-only log. They can be used to record events in the order they occur and then processed as a whole using complex consumption strategies such as consumer groups.
- Geospatial indexes are used for storing and searching location data in the form of coordinates. The most useful features include finding nearby points within a given radius or bounding box.

Redis Stack offers even more data types, including JSON, time series, and probabilistic data types like Bloom and Cuckoo filters.

Why is Redis fast?

- All Redis data resides in memory, which enables low latency and high throughput data access. Unlike traditional databases, in-memory data stores don't require slower disk access.
- Redis uses highly efficient data structures like skip-lists and simple dynamic strings.
- Redis can be deployed using a primary-replica architecture with support for asynchronous replication where data can be replicated to multiple

- Redis uses a single thread and I/O multiplexing to process requests, avoiding the cost of thread switching and lock resource contention between multiple threads.

Redis persistence

Redis is an in-memory data store, but some use cases also require persisting data to permanent disk storage. Redis provides multiple persistence options:

- No persistence: this option is used to disable persistence completely, usually when Redis is used as a cache.
- AOF (Append Only File): every write operation the server receives will be logged, and the logs can be used to reconstruct the original data if needed.
- RDB (Redis Database): performs point-in-time snapshots of the data at specified intervals.
- Hybrid (AOF + RDB): these two options can be combined.

RDB is a very compact single-file point-in-time representation of Redis data, perfect for backups. Compared to AOF, it allows faster restarts in the case of big datasets, but the RDB snapshots are usually created less frequently. Therefore, RDB persistence is not a good option if data loss is unacceptable in case Redis stops working. On the other side, AOF is more durable, with options to sync on every second or every query, usually resulting in bigger files than RDB. If a degree of data safety comparable to classic disk-based databases is desirable, a hybrid solution should be used. Constant disk writes can slow down Redis, so a compromise between performance and data safety needs to be made.

Factors impacting performance

- Redis is a server, meaning all commands involve network or IPC round trips. Network bandwidth and latency usually have a direct impact on the performance. Therefore, the lowest latencies are expected if the Redis server and client applications are in the same local network.
- As a single-threaded server, Redis favors fast CPUs with large caches and not many cores. If needed, several Redis instances should be used to scale out on several cores.
- Memory latency and bandwidth are less critical for the overall performance in the case of small objects. For objects larger than 10KB, it may become noticeable. Usually, it is not cost-effective to invest in expensive fast memory modules to optimize Redis.
- Redis is an in-memory data store with optional persistence options. Persistence comes at the cost of performance degradation. This should be taken into account when choosing a persistence strategy.
- Cache invalidation removes old data from cache, freeing up space and boosting the cache hit rate. This can be achieved manually or using automatically expiring keys. For every Redis key, a TTL (time to live) can be specified.

Typical use cases

After reviewing the main features of Redis, many use cases come to mind, including but not limited to caching. Here are only some of the more common ones:

- Database query results caching, persistent session caching, web page caching, and caching of frequently used objects such as images, files, and metadata are all popular examples of caching with Redis. Caching computationally expensive data or data that is a result of paid API calls is a good way to reduce costs.
- Rate limiting with Redis is easy, using keys with TTL and increment operation on values representing the access count.

intercommunication.

- Leaderboards can be implemented using the Redis Sorted Set data structure, which provides uniqueness of elements while maintaining the list sorted by scores.
- Location-based features such as drive time, drive distance, and points of interest using Redis built-in geospatial data structures.

Conclusion

Redis is a versatile in-memory data store that can be used in many use cases, including caching. Caching is a great option to improve performance and reduce cost but at the cost of increased system complexity. With its rich API, query support, a multitude of data structures, high availability, and clients for various programming languages, Redis has proven to be a viable option in most tech stacks. While Redis itself is fast and scales well, the performance gain it can bring to applications depends on multiple factors. Some of them are choosing the right architecture, cache invalidation strategy, frequency of data backups to permanent storage for a specific use case, and ensuring low network latencies when remote Redis instances are used. The most important decision is selecting data worth caching to ensure a high cache hit rate. In the end, a cache with a high hit rate is a successful cache.

“In-memory Caching using Redis” Tech Bite was brought to you by **Mustafa Šupuk**, Junior Software Engineer at Atlantbh.

Tech Bites are tips, tricks, snippets or explanations about various programming technologies and paradigms, which can help engineers with their everyday job.