gomomento.com

# Memcached vs. Redis: When scalability and reliability matter — Momento

*Eric O'Rearby Eric O'Rear, ,*

5-7 minutes

---

For organizations looking to produce highly scalable, distributed software, there are two principal in-memory data storage solutions to choose from: Memcached and Redis.

Memcached is a free and open source distributed memory object caching system. It provides an in-memory key-value store for data from sources such as database calls, API calls, and page rendering. Memory allocation with Memcached is flexible and efficient, making sure your system is well-provisioned regardless of shifting demands.

In recent years, Redis has taken the limelight as it has transitioned from a purely open source solution to a well-marketed, hybrid commercial offering. It is a datastore that can be used as a database, cache, and message broker, and provides support for a range of data structures and features.

While both are successful and useful tools in their own right, I want to focus on what makes Memcached such a great option: simplicity,

reliability, and scalability.

**Memcached is simple**

The true power of Memcached is its simplicity. Its distilled focus on providing high-performance caching of key-value objects, and not much else, keeps it lightweight and a go-to for developers and teams managing many potential points of failure.

Redis, on the other hand, offers support for an array of data types (strings, lists, sets, hashes, sorted sets, bitmaps, and more) and features (disk persistence, message brokerage, Lua scripting, transactions, pub/sub, and geospatial support)—if you need this degree of support from your caching solution, then Redis is for you!

But, if all you need to do is cache key-value objects, the simplest option is likely the best choice. In the complex world of distributed development and infrastructure, Memcached offers a much tighter set of features, resulting in a simpler tool for teams to use and manage.

**Memcached is reliable**

Memcached's singular data type and barebones set of features (its simplicity!) make it a more reliable tool with fewer failure modes than Redis. When and if something goes wrong in the caching layer, this simplicity makes finding the proverbial needle that much easier. A specific example of simplicity and reliability is how Memcached and Redis manage memory allocation differently. With Redis, the aim is to create a continuous block of used memory. As

data items are added, they are added “in a row” to previous data items. Need to remove an item? It will disappear from its place in the row, leaving a gap or hole in this memory block. This is called fragmentation. To deal with fragmentation, Redis runs operations to “clean up” memory allocation.

Memcached, on the other hand, prioritizes discrete blocks or chunks of memory for corresponding data items. If the chunk’s capacity is 80 bytes, and the data item is 60 bytes, there will be 20 unused bytes. Even if the next data item is 20 bytes, it will be stored in a discrete 80 byte chunk. While non-fragmented memory can be a negative from an efficiency point of view, it eliminates the need for additional background processes (and potential points of failure) that compact or rearrange stored data.

And this reliability is not just an issue of ‘solving the problem.’ Many enterprise customers of software come along with SLAs, or Service Level Agreements, involving contractual performance guarantees around things like latency and uptime. With the pivotal role of in-memory data storage in performant software, it is important to be able to predict in detail an implementation’s effects on system performance, lest teams find themselves in breach of SLAs.

**Memcached is scalable**

Simplicity and reliability are both strong reasons to scale (rather than replace) a software. With Memcached already a winner in the previous two categories, it may seem obvious that I’ve chosen it as the victor in scalability—and I have.

Redis has historically featured a single-core design, and required

users to launch and maintain a corresponding number of Redis instances to scale out onto several cores; but, as of Redis 6.0, Redis now supports multi-threading for I/O. So, depending on an implementation's versioning, Redis may require another extensive layer of management and provisioning for these additional instances, creating headaches for infrastructure teams trying to keep up with high velocity deployment schedules.

Memcached, on the other hand, supports multi-core machines, offering parallel processing across multiple threads on the same multi-core node. This means a given instance of Memcached can scale up and be given access to more CPU cores. Need more than one node? Memcached implementations allow applications to scale out dynamically, automatically increasing and decreasing nodes as system demand fluctuates.

Clustering is a common aspect of scaling, and while both systems offer cluster solutions, Memcached's cluster implementation is simpler, easier to set up, and less demanding to maintain.

**Conclusion**

Redis and Memcached are both excellent tools for in-memory data storage. But, if you don't need support for a range of data types, and don't need your in-memory solution to be responsible for performing a wide range of operations on the data it is storing, then Memcached will shine.

Trusted by high-load organizations such as Twitter, Netflix, and Facebook, Memcached's consistent emphasis on simplicity, be it in

its minimal feature set or in more technical decisions like memory allocation and cluster management, make it a reliable and scalable tool for caching in distributed systems.