# Memcached vs. Redis?

We're using a Ruby web-app with Redis server for caching. Is there a point to test Memcached instead?

What will give us better performance? Any pros or cons between Redis and Memcached?

Points to consider:

- Read/write speed.
- Memory usage.
- Disk I/O dumping.
- Scaling.

caching web-applications memcached redis

edited Aug 2 '16 at 10:57
Zeeshan Ali
66 ● 11

asked May 11 '12 at 20:52
Sagiv Ofek
11k ● 5 ● 37 ● 40

**protected** by Srikar Appalaraju Sep 23 '13 at 15:03

This question is protected to prevent "thanks!", "me too!", or spam answers by new users. To answer it, you must have earned at least 10 reputation on this site (the association bonus does not count).

18 Another analysis in addition to the below comments: Google Trends: redis vs. memcached – MarkHu Mar 27 '14 at 2:06

1 One comment that doesn't warrant an answer: if you're looking at cloud-based services for these two systems (e.g. heroku addons) Memcached services are sometimes quite a bit cheaper per MB for whatever reason. – Ben Roberts Nov 11 '14 at 19:22

1 For scalability: Imgur and Twitter use both – the_red_baron Nov 23 '16 at 0:38

17 Answers

## Summary (TL;DR)

**Updated June 3rd, 2017**

Redis is more powerful, more popular, and better supported than memcached. Memcached can only do a small fraction of the things Redis can do. Redis is better even where their features overlap.

For anything new, use Redis.

## Memcached vs Redis: Direct Comparison

Both tools are powerful, fast, in-memory data stores that are useful as a cache. Both can help speed up your application by caching database results, HTML fragments, or anything else that might be expensive to generate.

### Points to Consider

When used for the same thing, here is how they compare using the original question's "Points to Consider":

- **Read/write speed**: Both are extremely fast. Benchmarks vary by workload, versions, and many other factors but generally show redis to be as fast or almost as fast as memcached. I recommend redis, but not because memcached is slow. It's not.
- **Memory usage**: Redis is better.
  - memcached: You specify the cache size and as you insert items the daemon quickly grows to a little more than this size. There is never really a way to reclaim any of that space, short of restarting memcached. All your keys could be expired, you could flush the database, and it would still use the full chunk of RAM you configured it with.
  - redis: Setting a max size is up to you. Redis will never use more than it has to and will give you back memory it is no longer using.
  - I stored 100,000 ~2KB strings (~200MB) of random sentences into both. Memcached RAM usage grew to ~225MB. Redis RAM usage grew to ~228MB. After flushing both,

redis dropped to ~29MB and memcached stayed at ~225MB. They are similarly efficient in how they store data, but only one is capable of reclaiming it.

- **Disk I/O dumping**: A clear win for redis since it does this by default and has very configurable persistence. Memcached has no mechanisms for dumping to disk without 3rd party tools.
- **Scaling**: Both give you tons of headroom before you need more than a single instance as a cache. Redis includes tools to help you go beyond that while memcached does not.

### memcached

Memcached is a simple volatile cache server. It allows you to store key/value pairs where the value is limited to being a string up to 1MB.

It's good at this, but that's all it does. You can access those values by their key at extremely high speed, often saturating available network or even memory bandwidth.

When you restart memcached your data is gone. This is fine for a cache. You shouldn't store anything important there.

If you need high performance or high availability there are 3rd party tools, products, and services available.

### redis

Redis can do the same jobs as memcached can, and can do them better.

Redis can act as a cache as well. It can store key/value pairs too. In redis they can even be up to 512MB.

You can turn off persistence and it will happily lose your data on restart too. If you want your cache to survive restarts it lets you do that as well. In fact, that's the default.

It's super fast too, often limited by network or memory bandwidth.

If one instance of redis/memcached isn't enough performance for your workload, redis is the clear choice. Redis includes cluster support and comes with high availability tools (redis-sentinel) right "in the box". Over the past few years redis has also emerged as the clear leader in 3rd party tooling. Companies like Redis Labs, Amazon, and others offer many useful redis tools and services. The ecosystem around redis is much larger. The number of large scale deployments is now likely greater than for memcached.

## The Redis Superset

Redis is more than a cache. It is an in-memory data structure server. Below you will find a quick overview of things Redis can do beyond being a simple key/value cache like memcached. *Most* of redis' features are things memcached cannot do.

### Documentation

Redis is better documented than memcached. While this can be subjective, it seems to be more and more true all the time.

redis.io is a fantastic easily navigated resource. It lets you try redis in the browser and even gives you live interactive examples with each command in the docs.

There are now 2x as many stackoverflow results for redis as memcached. 2x as many Google results. More readily accessible examples in more languages. More active development. More active client development. These measurements might not mean much individually, but in combination they paint a clear picture that support and documentation for redis is greater and much more up-to-date.

### Persistence

By default redis persists your data to disk using a mechanism called snapshotting. If you have enough RAM available it's able to write all of your data to disk with almost no performance degradation. It's almost free!

In snapshot mode there is a chance that a sudden crash could result in a small amount of lost data. If you absolutely need to make sure no data is ever lost, don't worry, redis has your back there too with AOF (Append Only File) mode. In this persistence mode data can be synced to disk as it is written. This can reduce maximum write throughput to however fast your disk can write, but should still be quite fast.

There are many configuration options to fine tune persistence if you need, but the defaults are very sensible. These options make it easy to setup redis as a safe, redundant place to store data. It is a *real* database.

### Many Data Types

Memcached is limited to strings, but Redis is a data structure server that can serve up many different data types. It also provides the commands you need to make the most of those data types.

### Strings (commands)

Simple text or binary values that can be up to 512MB in size. This is the only data type redis and memcached share, though memcached strings are limited to 1MB.

Redis gives you more tools for leveraging this datatype by offering commands for bitwise operations, bit-level manipulation, floating point increment/decrement support, range queries, and multi-key operations. Memcached doesn't support any of that.

Strings are useful for all sorts of use cases, which is why memcached is fairly useful with this data type alone.

### Hashes (commands)

Hashes are sort of like a key value store within a key value store. They map between string fields and string values. Field->value maps using a hash are slightly more space efficient than key->value maps using regular strings.

Hashes are useful as a namespace, or when you want to logically group many keys. With a hash you can grab all the members efficiently, expire all the members together, delete all the members together, etc. Great for any use case where you have several key/value pairs that need to grouped.

One example use of a hash is for storing user profiles between applications. A redis hash stored with the user ID as the key will allow you to store as many bits of data about a user as needed while keeping them stored under a single key. The advantage of using a hash instead of serializing the profile into a string is that you can have different applications read/write different fields within the user profile without having to worry about one app overriding changes made by others (which can happen if you serialize stale data).

### Lists (commands)

Redis lists are ordered collections of strings. They are optimized for inserting, reading, or removing values from the top or bottom (aka: left or right) of the list.

Redis provides many commands for leveraging lists, including commands to push/pop items, push/pop between lists, truncate lists, perform range queries, etc.

Lists make great durable, atomic, queues. These work great for job queues, logs, buffers, and many other use cases.

### Sets (commands)

Sets are unordered collections of unique values. They are optimized to let you quickly check if a value is in the set, quickly add/remove values, and to measure overlap with other sets.

These are great for things like access control lists, unique visitor trackers, and many other things. Most programming languages have something similar (usually called a Set). This is like that, only distributed.

Redis provides several commands to manage sets. Obvious ones like adding, removing, and checking the set are present. So are less obvious commands like popping/reading a random item and commands for performing unions and intersections with other sets.

### Sorted Sets (commands)

Sorted Sets are also collections of unique values. These ones, as the name implies, are ordered. They are ordered by a score, then lexicographically.

This data type is optimized for quick lookups by score. Getting the highest, lowest, or any range of values in between is extremely fast.

If you add users to a sorted set along with their high score, you have yourself a perfect leader-board. As new high scores come in, just add them to the set again with their high score and it will re-order your leader-board. Also great for keeping track of the last time users visited and who is active in your application.

Storing values with the same score causes them to be ordered lexicographically (think alphabetically). This can be useful for things like auto-complete features.

Many of the sorted set commands are similar to commands for sets, sometimes with an additional score parameter. Also included are commands for managing scores and querying by score.

### Geo

Redis has several commands for storing, retrieving, and measuring geographic data. This includes radius queries and measuring distances between points.

Technically geographic data in redis is stored within sorted sets, so this isn't a truly separate data type. It is more of an extension on top of sorted sets.

### Bitmap and HyperLogLog

Like geo, these aren't completely separate data types. These are commands that allow you to treat string data as if it's either a bitmap or a hyperloglog.

Bitmaps are what the bit-level operators I referenced under `Strings` are for. This data type was the basic building block for reddit's recent collaborative art project: r/Place.

HyperLogLog allows you to use a constant extremely small amount of space to count almost unlimited unique values with shocking accuracy. Using only ~16KB you could efficiently count the number of unique visitors to your site, even if that number is in the millions.

### Transactions and Atomicity

Commands in redis are atomic, meaning you can be sure that as soon as you write a value to redis that value is visible to all clients connected to redis. There is no wait for that value to propagate. Technically memcached is atomic as well, but with redis adding all this functionality beyond memcached it is worth noting and somewhat impressive that all these additional data types and features are also atomic.

While not quite the same as transactions in relational databases, redis also has transactions that use "optimistic locking" (WATCH/MULTI/EXEC).

### Pipelining

Redis provides a feature called 'pipelining'. If you have many redis commands you want to execute you can use pipelining to send them to redis all-at-once instead of one-at-a-time.

Normally when you execute a command to either redis or memcached, each command is a separate request/response cycle. With pipelining, redis can buffer several commands and execute them all at once, responding with all of the responses to all of your commands in a single reply.

This can allow you to achieve even greater throughput on bulk importing or other actions that involve lots of commands.

### Pub/Sub

Redis has commands dedicated to pub/sub functionality, allowing redis to act as a high speed message broadcaster. This allows a single client to publish messages to many other clients connected to a channel.

Redis does pub/sub as well as almost any tool. Dedicated message brokers like RabbitMQ may have advantages in certain areas, but the fact that the same server can also give you persistent durable queues and other data structures your pub/sub workloads likely need, Redis will often prove to be the best and most simple tool for the job.

### Lua Scripting

You can kind of think of lua scripts like redis's own SQL or stored procedures. It's both more and less than that, but the analogy mostly works.

Maybe you have complex calculations you want redis to perform. Maybe you can't afford to have your transactions roll back and need guarantees every step of a complex process will happen atomically. These problems and many more can be solved with lua scripting.

The entire script is executed atomically, so if you can fit your logic into a lua script you can often avoid messing with optimistic locking transactions.

### Scaling

As mentioned above, redis includes built in support for clustering and is bundled with its own high availability tool called `redis-sentinel`.

## Conclusion

Without hesitation I would recommend redis over memcached for any new projects, or existing projects that don't already use memcached.

The above may sound like I don't like memcached. On the contrary: it is a powerful, simple, stable, mature, and hardened tool. There are even some use cases where it's a little faster than redis. I love memcached. I just don't think it makes much sense for future development.

Redis does everything memcached does, often better. Any performance advantage for memcached is minor and workload specific. There are also workloads for which redis will be faster, and many more workloads that redis can do which memcached simply can't. The tiny performance differences seem minor in the face of the giant gulf in functionality and the fact that both tools are so fast and efficient they may very well be the last piece of your infrastructure you'll ever have to worry about scaling.

There is only one scenario where memcached makes more sense: where memcached is already in use as a cache. If you are already caching with memcached then keep using it, if it meets your needs. It is likely not worth the effort to move to redis and if you are going to use redis just for caching it may not offer enough benefit to be worth your time. If memcached isn't meeting your needs, then you should probably move to redis. This is true whether you need to scale beyond memcached or you need additional functionality.

edited Jun 3 at 7:08 answered Jun 29 '12 at 6:54

Carl Zulauf
**28.9k** ● 2 ● 21 ● 38

7 How does Memcached offer clustering in a way that exists in the server themselves? I've always used libraries that distributed to a pool of memcached servers using hashing algorithms or a modulus. The same is said for Redis. I mostly use Python and there seem to be quite a few modules that don't rely on the memcached library to handle connection pools. – whardier Oct 9 '12 at 4:57

1 "Transactions with optimistic locking (WATCH/MULTI/EXEC)" - Redis has no right transactions. I.e. if [multi, cmd1, cmd2, cmd3 (exception) , exec] then cmd1 and cmd2 will be executed. – Oleg Feb 20 '13 at 12:40

7 @Oleg that is not actually true. If you use multi-exec the commands are buffered (ie: not executed) until the exec occurs, so if you have an exception before the exec then no commands are actually executed. If exec is called all the buffered commands are executed atomically, unless, of course, a watch variable has been changed since multi was first called. This latter mechanism is the optimistic locking part. – Carl Zulauf Mar 20 '13 at 1:47

3 @whardier You're correct. Updated answer to reflect that memcached's cluster "support" is enabled by additional tools. Should have researched that better. – Carl Zulauf Apr 13 '13 at 22:20

2 how about clustering with couchbase server? (memcached compatible) – Ken Liu Mar 6 '14 at 15:33

|

---

Use Redis if

1. You require selectively deleting/expiring items in the cache. (You need this)
2. You require the ability to query keys of a particular type. eq. 'blog1:posts:*', 'blog2:categories:xyz:posts:*'. oh yeah! this is very important. Use this to invalidate certain types of cached items selectively. You can also use this to invalidate fragment cache, page cache, only AR objects of a given type, etc.
3. Persistence (You will need this too, unless you are okay with your cache having to warm up after every restart. Very essential for objects that seldom change)

Use memcached if

1. Memcached gives you headached!
2. umm... clustering? meh. if you gonna go that far, use Varnish and Redis for caching fragments and AR Objects.

From my experience I've had much better stability with Redis than Memcached

answered Jul 5 '12 at 6:04

SMathew
**3,562** ● 1 ● 12 ● 10

7 Redis documentation says that using patterns requires a table scan. blog1:posts:* may require an O(N) table scan. Of course, it's still fast on reasonably sized data sets, since Redis is fast. It should be OK for testing or admin. – wisty Nov 1 '12 at 7:44

125 *Headached* is a joke, right? :-) I googled for *memcached headached* but didn't find anything reasonable. (I'm new to Memcached and Redis) – KajMagnus Jul 31 '13 at 8:42

5 voted *down* for the same reason than @pellucide. Redis might be better than Memcached, but Memcached is trivial to use. I never had a problem with it and it's trivial to configure. – Diego Jancic Jul 30 '15 at 13:24

1 Thank you @KajMagnus for making my day.. possibly my entire week 😂 – alex Apr 29 at 1:33

---

Memcached is multithreaded and fast.

Redis has lots of features and is very fast, but completely limited to one core as it is based on an event loop.

We use both. Memcached is used for caching objects, primarily reducing read load on the databases. Redis is used for things like sorted sets which are handy for rolling up time-series data.

answered May 3 '13 at 16:41

W. Andrew Loe III
**1,138** ● 10 ● 9

2 High-traffic sites that are heavily invested in memcached and have db bottlenecks on "user profile"-like non-relational data should evaluate couchbase in parallel with the usual Mongo, Redis – Barry Feb 28 '15 at 23:47

1 I believe the 'limited to one core' is no longer an issue with Redis 3, though I could be wrong. – siliconrockstar Jan 12 at 19:43

---

**This is too long to be posted as a comment to already accepted answer, so I put it as a**

**separate answer**

One thing also to consider is whether you expect to have a hard upper memory limit on your cache instance.

Since redis is an nosql database with tons of features and caching is only one option it can be used for, it allocates memory as it needs it — the more objects you put in it, the more memory it uses. The `maxmemory` option does not strictly enforces upper memory limit usage. As you work with cache, keys are evicted and expired; chances are your keys are not all the same size, so internal memory fragmentation occurs.

By default redis uses jemalloc memory allocator, which tries its best to be both memory-compact and fast, but it is a general purpose memory allocator and it cannot keep up with lots of allocations and object purging occuring at a high rate. Because of this, on some load patterns redis process can apparently leak memory because of internal fragmentation. For example, if you have a server with 7 Gb RAM and you want to use redis as non-persistent LRU cache, you may find that redis process with `maxmemory` set to 5Gb over time would use more and more memory, eventually hitting total RAM limit until out-of-memory killer interferes.

memcached is a better fit to scenario described above, as it manages its memory in a completely different way. memcached allocates one big chunk of memory — everything it will ever need — and then manages this memory by itself, using its own implemented slab allocator. Moreover, memcached tries hard to keep internal fragmentation low, as it actually uses per-slab LRU algorithm, when LRU evictions are done with object size considered.

With that said, memcached still has a strong position in environments, where memory usage has to be enforced and/or be predictable. We've tried to use latest stable redis (2.8.19) as a drop-in non-persistent LRU-based memcached replacement in workload of 10-15k op/s, and it leaked memory A LOT; the same workload was crashing Amazon's ElastiCache redis instances in a day or so because of the same reasons.

edited Mar 2 '15 at 11:22

answered Mar 2 '15 at 11:13

artyom
962 ● 8 ● 17

From redis.io/topics/faq: *Redis has built-in protections allowing the user to set a max limit to memory usage, using the maxmemory option in the config file to put a limit to the memory Redis can use. If this limit is reached Redis will start to reply with an error to write commands (but will continue to accept read-only commands), or you can configure it to evict keys when the max memory limit is reached in the case you are using Redis for caching. We have documentation if you plan to use Redis as an LRU cache.* link – StefanNch Sep 4 '15 at 8:11

4 @StefanNch redis' `maxmemory` option does not account for internal memory fragmentation. Please see my comment above for details — the problems I've described there were seen under the scenario described in "Redis as an LRU cache" page with memory limiting options enabled. memcached, on the other side, uses different approach to avoid memory fragmentation problem, so its memory limit is much more "hard". – artyom Sep 7 '15 at 15:42

---

Memcached is good at being a simple key/value store and is good at doing key => STRING. This makes it really good for session storage.

Redis is good at doing key => SOME_OBJECT.

It really depends on what you are going to be putting in there. My understanding is that in terms of performance they are pretty even.

Also good luck finding any objective benchmarks, if you do find some kindly send them my way.

edited May 11 '12 at 23:32

answered May 11 '12 at 23:27

Erik Petersen
1,939 ● 7 ● 15

2 IMO the Redis Hash data type makes a lot more sense for storing session variables than serializing them into a memcached string. – Carl Zulauf Jun 29 '12 at 7:28

5 If you care about user experience, do not put your sessions in cache. dormando.livejournal.com/495593.html – sleblanc Mar 22 '13 at 4:17

3 @sebleblanc This shouldn't theoretically be an issue with Redis however since there is disk persistency as well. – haknick Nov 14 '13 at 19:04

2 @sebleblanc memcache is still good at session storage you implement it poorly or not. yes eviction is a problem but not in anyway insurmountable, also it is not memcache's problem if you don't worry about eviction. Most memcache session solutions use cookies as a backup I believe. – Erik Petersen Dec 9 '13 at 1:51

6 "Do not put your sessions in cache" is misleading. What you mean is "Do not only store your sessions in cache". Anyone who stores important data in memcache only should be fired immediately. – Jacob Jul 9 '14 at 4:38

|

---

If you don't mind a crass writing style, Redis vs Memcached on the Systoilet blog is worth a read from a usability standpoint, but be sure to read the back & forth in the comments before

drawing any conclusions on performance; there are some methodological problems (single-threaded busy-loop tests), and Redis has made some improvements since the article was written as well.

And no benchmark link is complete without confusing things a bit, so also check out some conflicting benchmarks at Dormondo's LiveJournal and the Antirez Weblog.

*Edit* -- as Antirez points out, the Systoilet analysis is rather ill-conceived. Even beyond the single-threading shortfall, much of the performance disparity in those benchmarks can be attributed to the client libraries rather than server throughput. The benchmarks at the Antirez Weblog do indeed present a much more apples-to-apples (with the same mouth) comparison.

edited Jan 3 '13 at 14:02

answered Jun 15 '12 at 22:38

Paul Smith
**1,627** ● 21 ● 33

9 the Redis vs Memcached benchmark is ill conceived.oldblog.antirez.com/post/redis-memcached-benchmark.html – App Work Dec 30 '12 at 7:37

21 You weren't kidding about crass. – ocodo Jan 3 '13 at 0:59

More over its 2010, outdated blog – Siddharth Sep 18 '15 at 13:23

---

I got the opportunity to use both memcached and redis together in the caching proxy that i have worked on , let me share you where exactly i have used what and reason behind same....

Redis >

1) Used for indexing the cache content , over the cluster . I have more than billion keys in spread over redis clusters , redis response times is quite less and stable .

2) Basically , its a key/value store , so where ever in you application you have something similar, one can use redis with bothering much.

3) Redis persistency, failover and backup (AOF ) will make your job easier .

Memcache >

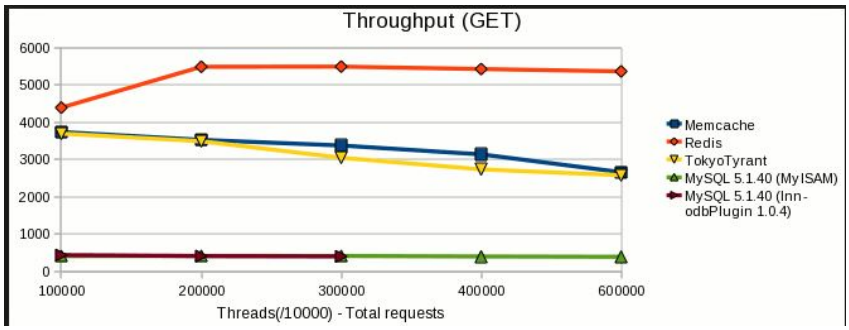
1) yes , an optimized memory that can be used as cache . I used it for storing cache content getting accessed very frequently (with 50 hits/second)with size less than 1 MB .

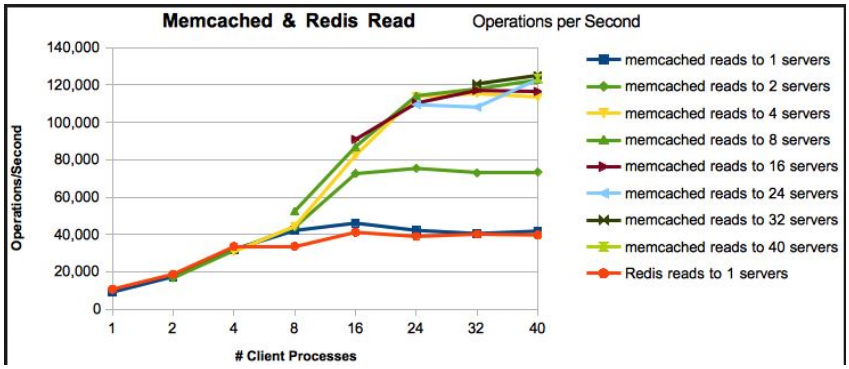
2) I allocated only 2GB out of 16 GB for memcached that too when my single content size was >1MB .

3) As the content grows near the limits , occasionally i have observed higher response times in the stats(not the case with redis) .

If you ask for overall experience Redis is much green as it is easy to configure, much flexible with stable robust features.

Further , there is a benchmarking result available at this link , below are few higlight from same,



Hope this helps!!

edited Jun 29 '16 at 6:57

answered Oct 16 '15 at 10:20

Jain Rach

**1,618** ● 1 ● 8 ● 19

---

Another bonus is that it can be very clear how memcache is going to behave in a caching scenario, while redis is generally used as a persistent datastore, though it can be configured to behave just like memcached aka evicting Least Recently Used items when it reaches max capacity.

Some apps I've worked on use both just to make it clear how we intend the data to behave - stuff in memcache, we write code to handle the cases where it isn't there - stuff in redis, we rely on it being there.

Other than that Redis is generally regarded as superior for most use cases being more feature-rich and thus flexible.

answered Jul 4 '12 at 12:42

Scott Schulthess

**1,760** ● 2 ● 20 ● 31

---

Test. Run some simple benchmarks. For a long while I considered myself an old school rhino since I used mostly memcached and considered Redis the new kid.

With my current company Redis was used as the main cache. When I dug into some performance stats and simply started testing, Redis was, in terms of performance, comparable or minimally **slower** than MySQL.

Memcached, though simplistic, blew Redis out of water **totally**. It scaled much better:

- for bigger values (required change in slab size, but worked)
- for multiple concurrent requests

Also, memcached eviction policy is in my view, much better implemented, resulting in overall more stable average response time while handling more data than the cache can handle.

Some benchmarking revealed that Redis, in our case, performs very poorly. This I believe has to do with many variables:

- type of hardware you run Redis on
- types of data you store
- amount of gets and sets
- how concurrent your app is
- do you need data structure storage

Personally, I don't share the view Redis authors have on concurrency and multithreading.

answered Nov 13 '15 at 8:14

mdomans

**1,127** ● 10 ● 16

---

One major difference that hasn't been pointed out here is that Memcache has an upper memory limit at all times, while Redis does not by default (but can be configured to). If you would always like to store a key/value for certain amount of time (and never evict it because of low memory) you want to go with Redis. Of course, you also risk the issue of running out of memory...

answered Mar 1 '13 at 9:45

Ztyx

**4,087** ● 4 ● 31 ● 63

---

It would not be wrong, if we say that redis is combination of (cache + data structure) while memcached is just a cache.

answered Jun 16 '15 at 5:45

Atif Hussain

**194** ● 1 ● 12

this is good answer - Laravel is using redis as cache and as data storage mechanism – carousel Jul 26 '15 at 17:42

---

We thought of Redis as a load-takeoff for our project at work. We thought that by using a module in nginx called HttpRedis2Module or something similar we would have awesome speed but when testing with AB-test we're proven wrong.

Maybe the module was bad or our layout but it was a very simple task and it was even faster to take data with php and then stuff it into MongoDB. We're using APC as caching-system and with that php and MongoDB. It was much much faster then nginx Redis module.

My tip is to test it yourself, doing it will show you the results for your environment. We decided that using Redis was unnecessary in our project as it would not make any sense.

answered Jul 5 '12 at 11:48

Markus Stenqvist
**1,435** ● 4 ● 27 ● 61

2 caching was slower then db queries? sounds weird.. – Sagiv Ofek Jul 5 '12 at 12:46

Interesting answer but not sure if it helps out the OP – Scott Schulthess Jul 5 '12 at 17:40

Inserting to Redis and using it as cache was slower than using APC + PHP + MongoDB. But just the insertion to Redis was MUCH slower than inserting directly into MongoDB. Without APC I think they're pretty equal. – Markus Stenqvist Jul 6 '12 at 9:59

2 Thats because mongo doesn't give you any guarantee that what you've inserted is *ever* going to be written to disk... – Damian Apr 24 '14 at 7:01

15 but it is webscale, mongodb will run around you in circles while you write. Nowadays I only write to /dev/null because that is the fastest. – Markus Stenqvist Apr 24 '14 at 19:27

---

The biggest remaining reason is specialization.

Redis can do a lot of different things and one side effect of that is developers may start using a lot of those different feature sets on the same instance. If you're using the LRU feature of Redis for a cache along side hard data storage that is NOT LRU it's entirely possible to run out of memory.

If you're going to setup a dedicated Redis instance to be used ONLY as an LRU instance to avoid that particular scenario then there's not really any compelling reason to use Redis over Memcached.

If you need a reliable "never goes down" LRU cache...Memcached will fit the bill since it's impossible for it to run out of memory by design and the specialize functionality prevents developers from trying to make it so something that could endanger that. Simple separation of concerns.

answered Nov 20 '15 at 21:32

aramisbear
**612** ● 6 ● 9

---

Redis is better The Pros of Redis are ,

```
1.It has a lot of data storage options such  as  string  , sets , sorted sets ,
hashes ,  bitmaps
2.Disk Persistence of records
3.Stored Procedure (LUA acripting)  support
4.Can act as a Message Broker using PUB/SUB
```

Whereas Memcache is an in-memory key value cache type system.

1. No support for various data type storages like lists , sets as in redis.
2. The major con is Memcache has no disk persistence .

answered Mar 22 at 4:22

Athavan Kanapuli
**107** ● 1 ● 10

---

I'd say redis, as it's much faster than memcached, even though it runs on a single core of a machine, and memcached can be run in parallel, it'd take a lot of processing to hit redis's upper limit when configured correctly

answered Mar 12 '15 at 10:36

serdarsenay
**502** ● 4 ● 13

---

Well I mostly used both with my apps, Memcache for cache the sessions and redis for doctrine/orm queries objects. In terms of performance both are almost same.

answered Oct 31 '16 at 21:08

mtaqi

1,509 ● 2 ● 17 ● 34

---

A very simple test to set and get 100k unique keys and values against redis-2.2.2 and memcached. Both are running on linux VM(CentOS) and my client code(pasted below) runs on windows desktop.

**Redis** *Time taken to store 100000 values is = 18954ms Time taken to load 100000 values is = 18328ms*

**Memcached** *Time taken to store 100000 values is = 797ms Time taken to retrieve 100000 values is = 38984ms*

```
Jedis jed = new Jedis("localhost", 6379);
int count = 100000;
long startTime = System.currentTimeMillis();
for (int i=0; i<count; i++) {
jed.set("u112-"+i, "v51"+i);
}
long endTime = System.currentTimeMillis();
System.out.println("Time taken to store "+ count + " values is ="+(endTime-
startTime)+"ms");

startTime = System.currentTimeMillis();
for (int i=0; i<count; i++) {
client.get("u112-"+i);
}
endTime = System.currentTimeMillis();
System.out.println("Time taken to retrieve "+ count + " values is ="+(endTime-
startTime)+"ms");
```

answered Jun 1 at 6:36

Prabhu Nandan Kumar

368 ● 1 ● 10

---