**CACHES COMPARISATIONS:**

**Alcuni pensieri: (da verificare)**

**Uso concorrente CPU**<http://stackoverflow.com/questions/10558465/memcached-vs-redis>

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.

EXP:  
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.

**Consumo memoria:**<http://stackoverflow.com/questions/10558465/memcached-vs-redis>

**Sintesi:**

Redis non ha limiti nel consumo della memoria.  
Ne consuma di piu offrendo molti altri servizi (MOM, persistenza, etc)  
Usa jemalloc che per alti volume in cancellazione/scrittura frammenta la memoria, rendendo piu lenta le lettura successiva.

Memcached usa un grosso chunk di memoria che alloca all’inizio (e non si espande piu?) nel quale fa stare tutto.  
Usa uno slaballocator che riesce a gestire la memoria in modo piu compatto anche sotto alto carico.  
  
Benckmakrs devono considerare sempre uso di memoria in situazioni con alti livelli di cancellazione/scrittura nuovi dati.

**Quote:**

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

REDIS:  
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](http://www.canonware.com/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

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](https://en.wikipedia.org/wiki/Slab_allocation).   
Moreover, memcached tries hard to keep internal fragmentation low, as it actually [uses per-slab LRU algorithm](https://www.adayinthelifeof.nl/2011/02/06/memcache-internals/), when LRU evictions are done with object size considered.

**How to benckmark**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.

Commento:  
*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.*

**Consumo memoria:**

Memcached supporta solo String. Per cui consuma meno spazio per i metadata.

**Cache o accesso a disco prevebidile: (gestita manualmente/automaticamente)**

**Sintesi**

Con alcune configurazioni, Redis nel caso non trovasse un entry nella cache lo cerca nel livello di persistenza.  
Memcache non avendo un livello di persistenza lascia allo sviluppatore questa decisione/incombenza.

**Quote**:

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.

**HORIZONTAL SCALING:**

Piu facile scalare su N server Memcached. Redis e’ piu legato al Master/Slave model.  
Anche se dalla versione 3 in poi, ci sono dei workaround.

**EVICTION POLICIES:**

Memcached supporta solo LRU (richiesto da piu tempo).  
In modo piu o meno arbitrario fa un evict di un entry con lo stesso spazio di quella da caricare cercandola tra quelle che hanno avuto l’utlima richiesta molto tempo fa.  
  
Redis, permette di configurare 6 diversi modelli di eviction.

**Tipi di dati supportati:**

Whereas Memcached limits key names to 250 bytes, limits values to 1MB, and works only with plain strings, Redis allows key names and values to be as large as 512MB each, and they are binary safe

Whereas Memcached limits key names to 250 bytes, limits values to 1MB, and works only with plain strings,

Redis allows key names and values to be as large as 512MB each, and they are binary safe.   
  
Redis has six data types that enable more intelligent caching and manipulation of cached data, opening up a world of possibilities to the application developer.  
  
Instead of storing objects as serialized strings, the developer can use a Redis Hash to store an object's fields and values and manage them using a single key. Redis Hash saves developers the need to fetch the entire string, de-serialize it, update a value, re-serialize the object, and replace the entire string in the cache with its new value for every trivial update -- and that means lower resource consumption and increased performance.   
Other data types that Redis offers, such as Lists and Sets, can be leveraged to implement even more complex cache management patterns.

Possibilita di processare tramite script I dati direttamente su Redis.

Another important advantage of Redis is that the data it stores isn't opaque, meaning that the server can manipulate it directly. A considerable share of the 160-plus commands available in Redis is devoted to data processing operations and embedding logic in the data store itself via server-side scripting.

These built-in commands and user scripts give you the flexibility of handling data processing tasks directly in Redis, without having to ship data across the network to another system for processing.

REDIS PERISTENZA:

Redis offers optional and tunable data persistence, which is designed to bootstrap the cache after a planned shutdown or an unplanned failure.   
While we tend to regard the data in caches as volatile and transient, persisting data to disk can be quite valuable in caching scenarios. Having the cache's data available for loading immediately after restart allows for much shorter cache warm-up periods and removes the load involved in repopulating and recalculating cache contents from the primary data store.

REDIS REPLICATION

Last but not least, Redis offers replication. Replication can be used for implementing a highly available cache setup that can withstand failures and provide uninterrupted service to the application. Considering a cache failure falls only slightly short of application failure in terms of the impact on user experience and application performance, having a proven solution that guarantees the cache's contents and service availability is a major advantage in most cases.

**Performance**:

Redis is faster per core with small values, but memcached is able to use multiple cores with a single executable and TCP port without help from the client. Also memcached is faster with big values in the order of 100k. Redis recently improved a lot about big values (unstable branch) but still memcached is faster in this use case. The point here is: nor one or the other will likely going to be your bottleneck for the query-per-second they can deliver.

**Set of operations:**

Redis molto ricco

**REDIS HAZELCAST:**

**Tipi dati salvati:**

Redis: String, Hash, List, Set, SortedSet  
Hazelcast: Jva Collections  
Resisson (libreria per Redis) aggiunge java collections ai tipi redis.

### Persistence in Hazelcast and Redis

Redis supports writing of data to disk automatically with snapshotting and append-only files.   
Hazelcast uses MapLoader and MapStore interfaces for flexible persistence. Also, Hazelcast can store data in any underlying storage, including Redis itself.

**Limitazioni Redis dati:**

You can do "almost" the same operations with strings, lists, sets as you usually do working with simple String, ArrayList or HashSet in Java. Unfortunately, "almost" means not really, pardon the pun. For example, there is no ability to check whether the list contains some element or not. Strange? Yes, it's very strange. You wanna another "funny" example? Well, it's simple - you can get a range of elements from the sorted set but cannot do that using a regular one. Let's imagine that you don't want to use different data structures and want to use Redis as a simple key/value store, like a simple map. Another surprises will appear, be sure. For example it's not so simple to iterate over this map or get the ranges. I would say that it's even impossible to do without creating an additional key where you will store a list or a set of keys. What do you think about ability to specify the DB not by name and just by index? Why it's like that? - I have no idea.

Couchbase

Then we were trying to see how both data buckets provided by Couchbase are working. From the first look Couchbase is awesome. It has a lot of different options like clustering, auto-failover, map/reduce and persistence for Couchbase buckets,*asynchronous operations* and many other things. Sounds good, yeah? Then we run our test and it quickly put everything in its place. The Couchbase Admin page showed incredible numbers for remote connection 50 000-60 000 ops/sec and >100 000 ops/sec for local connection, however only 40-60% of all data we were trying to load to the bucket was there in the end of the test. All other operations failed because of timeout. The Couchbase experts explained it by the fact that we didn't check the result of set/add operations. Yes, it was true, because when you add this check it means you are using synchronous operations instead of asynchronous and performance degradation is very impressive (<1000 ops/sec for remote connection and 20 000 - 30 000 ops/sec for local connection). So, taking into account that our distributed cache must handle huge amount of operations during bulk load correctly and in a very fast way, we decided to find luck somewhere else.

**TEST PER BENCKMARKS:**

**Write:**Salvare 10.000.000 di oggetti nel sistema.

**Read:**recuperare oggetti salvati

**Write/Delete:**salva/expire 10.000.000 oggetti