In-Process Caching vs. Distributed Caching

https://dzone.com/articles/process-caching-vs-distributed

As the name suggests, an in-process cache is an object cache built within the same address space as your application. The Google Guava Library provides a simple in-process cache API that is a good example. On the other hand, a distributed cache is external to your application and quite possibly deployed on multiple nodes forming a large logical cache. Memcached is a popular distributed cache.Ehcache from Terracotta is a product that can be configured to function either way.

Following are some considerations that should be kept in mind when making a decision.

Considerations	In-Process Cache	Distributed Cache	Comments
Consistency	While using an in-process cache, your cache elements are local to a single instance of your application. Many medium-to-large applications, however, will not have a single application instance as they will most likely be load-balanced. In such a setting, you will end up with as many caches as your application instances, each having a different state resulting in inconsistency. State may however be eventually consistent as cached items time-out or are evicted from all cache instances.	Distributed caches, although deployed on a cluster of multiple nodes, offer a single logical view (and state) of the cache. In most cases, an object stored in a distributed cache cluster will reside on a single node in a distributed cache cluster. By means of a hashing algorithm, the cache engine can always determine on which node a particular key-value resides. Since there is always a single state of the cache cluster, it is never inconsistent.	If you are caching immutable objects, consistency ceases to be an issue. In such a case, an in-process cache is a better choice as many overheads typically associated with external distributed caches are simply not there. If your application is deployed on multiple nodes, you cache mutable objects and you want your reads to always be consistent rather than eventually consistent, a distributed cache is the way to go.
Overheads	This dated but very descriptive article describes how an in-process cache can negatively effect performance of an application with an embedded cache primarily due to garbage collection overheads. Your results however are heavily dependent on factors such as the size of the cache and how quickly objects are being evicted and timed-out.	A distributed cache will have two major overheads that will make it slower than an in-process cache (but better than not caching at all): network latency and object serialization	As described earlier, if you are looking for an always-consistent global cache state in a multi-node deployment, a distributed cache is what you are looking for (at the cost of performance that you may get from a local in-process cache).
Reliability	An in-process cache makes use of the same heap space as your program so one has to be careful when determining the upper limits of memory usage for the cache. If your program runs out of memory there is no easy way to recover from it.	A distributed cache runs as an independent processes across multiple nodes and therefore failure of a single node does not result in a complete failure of the cache. As a result of a node failure, items that are no longer cached will make their way into surviving nodes on the next cache miss. Also in the case of distributed caches, the worst	An in-process cache seems like a better option for a small and predictable number of frequently accessed, preferably immutable objects. For large, unpredictable volumes, you are better off with a distributed cache.

		consequence of a complete cache failure should be degraded performance of the application as opposed to complete system failure.	

Recommendation

For a small, predictable number of preferably immutable objects that have to be read multiple times, an in-process cache is a good solution because it will perform better than a distributed cache. However, for cases in which the number of objects that can be or should be cached is unpredictable and large, and consistency of reads is a must-have, a distributed cache is perhaps a better solution even though it may not bring the same performance benefits as an in-process cache. It goes without saying that your application can use both schemes for different types of objects depending on what suits the scenario best.