Caching Strategies

View Source: Design Patterns in the Wild

© Luc De Leeuw
Today

- Memory Caching
- Distributed Caching
- Layered Cache Architectures
- Abstract Factory Pattern
Why Cache?

• Store data closest to where it's needed
• Depending on where data is stored, trade-off between:
  – CPU cycles
  – Memory consumption
  – Disk access
  – Network utilization
Cache Storage

- Increase locality of data
- Balance resource usage for data retrieval and storage
- Remember – it's a temporary store
Caching Strategies

- **Cost-based**
  - Fixed size
  - Responsive to item state
- **Memory sensitive**
  - Variable size
  - Responsive to system state
- **Time sensitive**
  - Often combined with other strategies
Hit Ratio

\[
\text{Hits} = \frac{\text{Hits}}{\text{Total Requests}}
\]

Hits - Requests which find items in cache
Cost-based Caching

- Fixed cache size
- Cost-based algorithm to maintain size
  - LRU, LFU, etc.
- Cache size critical to performance
Abstract Factory

AbstractFactory

+makeWidget(): Widget
+getInstance(): AbstractFactory

BarFactory

+makeWidget(): Widget

FooFactory

+makeWidget(): Widget

Widget

BarWidget

FooWidget
Abstract Factory in Hibernate

CacheFactory

+createCache(concurrencyStrategy: String,
regionName: String, mutable: boolean,
settings: Settings, properties: Properties): CacheConcurrencyStrategy

Settings

+getCacheProvider(): CacheProvider

CacheProvider

+buildCache(regionName: String, properties: Properties): Cache

OSCacheProvider

+buildCache(regionName: String, properties: Properties): Cache

SwarmCacheProvider

+buildCache(regionName: String, properties: Properties): Cache

Cache

OSCache

SwarmCache
Using Abstract Factory

- Many products, many factories
- Delegate construction to different factory method implementations
- Flexibility to configure in new products and associated factory implementations
Soft References

- Java's Garbage Collector frees up memory by removing unreferenced objects
- Soft References are maintained as long as the GC is not seeking memory
Memory Sensitive Caching

- Variable size
- Responsive to system constraints
- Potential object churn under load
Stacking Strategies

SoftLimitMRUCache
- softReferenceCache: ReferenceMap
- strongReferenceCache: LRUMap
+ get(key:Object): Object
+ put(key:Object, value:Object)

ReferenceMap
+ object(key:Object): get
+ put(key:Object, value:Object)

LRUMap
+ object(key:Object): get
+ put(key:Object, value:Object)
Stacking with Thrudb


Ashwin J Mathew          Spring 2008
School of Information, UC Berkeley
Using Caching

• **Improve performance**
  – Increase locality of data
  – Pre-process data

• **Tuning is critical**
  – Hit Ratio
  – Overall system performance

• **Warning:** Caches may sync amongst themselves – but yet be out of sync with underlying data stores