INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2022-06-09

EXAMENSARBETE Database loading strategies for an in-memory cache in Java
STUDENTER Ivar Henckel, David Söderberg
HANDLEDARE Alma Orucevic-Alagic (LTH), Inger Klintmalm (Nasdaq)
EXAMINATOR Jonas Skeppstedt (LTH)

Loading data into an in-memory cache, the choice of being full, lazy, or asynchronous

POPULÄRVETENSKAPLIG SAMMANFATTNING **Ivar Henckel, David Söderberg**

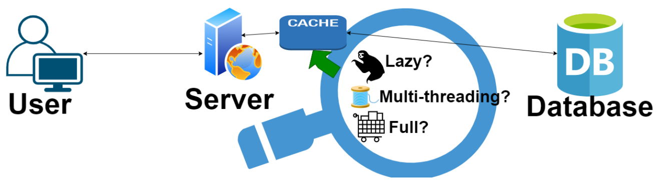
Slow database loading can be a nuisance, causing unresponsive programs and irritated customers. In our thesis, we investigate different strategies to load data into a cache efficiently. We propose new implementations, some of which manage to decrease the inconvenience of data loading.

In the field of computer science databases are commonly used as a means to effectively store and fetch data. Although databases are optimized for fetching data with the high performance there is still a cost for each load, especially with a huge amount of data. To mitigate this cost caches can be used to temporarily store previously accessed data in a location that is faster to access.

In this thesis, we investigated different strategies to load data into the cache. The original cache implementation at the case company loaded all of the data into the cache at startup, called a *full load*. This meant that startup time would increase which could be a problem, especially if a server crashes during business hours and everything needs to be quickly restarted and restored.

One alternative solution is to load data into the cache *lazily*. Lazy loading means that no load is done until the data is used in the program. This will move the cost of loading from startup to run-time while also removing the cost of fetching data that is never used. A third option is to utilize parallelism and load data from separate threads while the main thread can continue without having to wait.

Three solutions were implemented, two of which used lazy loading, and the third used parallelism. These implementations were compared to the original full load solution in experiments. It was found that all three implementations decreased startup time but impacted the run-time request latency in different ways. Additionally, all of our solutions added a higher level of complexity to a varying degree, compared to the full load solution. One of the lazy loading implementations was shown to be inefficient. This solution relied heavily on a higher level framework meaning that we as developers had less control over the details in the database operations. The second lazy loading solution worked efficiently, serving its purpose to reduce startup time but slightly increase run-time latency as expected. Our implementation using parallelism was shown to be efficient both at startup and run-time, but it also adds the most complexity.


User
Server
CACHE
Lazy?
Multi-threading?
Full?
DB
Database