NSS - Cache

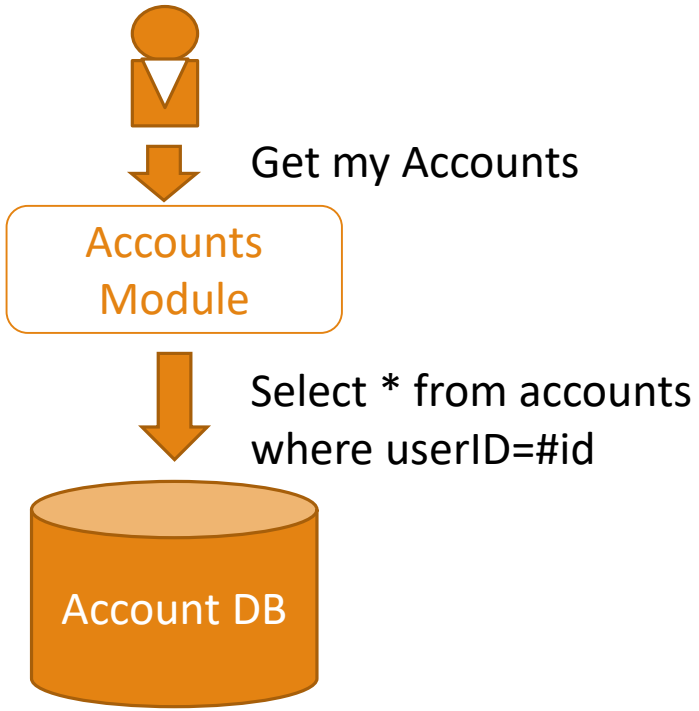
4. EXERCISE

LUKAS BARTONEK

Kdy cachovat

1. Pokud se načtená data znovu přepoužívají
2. Zrychlení
    1. kolikrát jsou načtená data znovupoužita aplikací
    2. jak moc se zmenší čas odpovědi při použití cachování

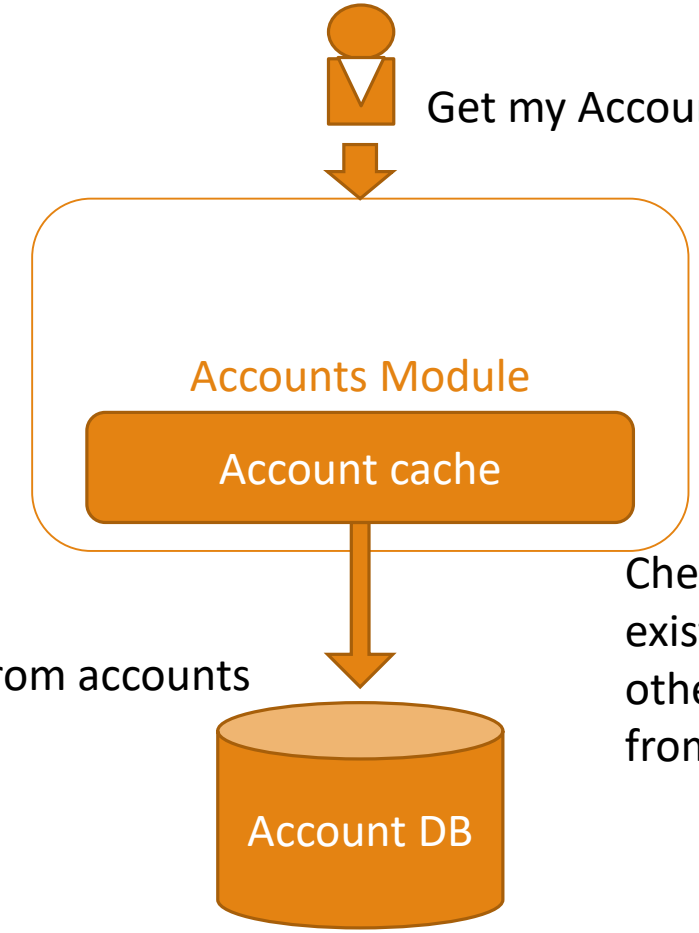
Ukázka využití cache



Počet uživatelů 1 000 000
Response time modulu (bez DB): 100 ms
Response time DB: 20ms

Počet volání Modulu	Počet volání DB	Celkový počet volání	Celkový čas (hodiny)
1 000 000	1 000 000	2 000 000	33

Ukázka využití cache



Check data in cache, if exists return it, otherwise také all data from DB

Počet uživatelů 1 000 000
Response time modulu (bez DB): 100 ms
Response time DB: 20ms
Response time cache: 3ms

Počet volání Modulu	Počet volání DB	Počet volání cache	Celkový počet volání	Celkový čas (hodiny)
1 000 000	1	1 000 000	2 000 001	28,6

Lze využít

1. Optimalizaci výkonu systému
2. Snížení náročností jednotlivých operací
3. Snížení náročností na jednotlivé vrstvy
4. Minimalizace rizika krátkodobé nedostupnosti systému

Node cache

1. Jediný node – který obsahuje všechna cachovaná data
    1. Ekvivalent Clustrové cache s jedním membrem
2. Například JBOSS TreeCache

Úvod do praktické ukázky

Máte 2 servery, každý tento server má nadeployované WAR, které lze provolat.

Toto WAR provolává třídu XYZ, která následně získává data z třídy YYY. Bohužel třída YYY je velmi pomalá (kromě třídy se může jednat i o další systém, který nemáte pod kontrolou) a tudíž jsou pomalá i vaše volání třídy XYZ.

Data v třídy YYY jsou statického charakteru a mění se jednou za X.

V ideálním světe změníme chování třídy YYY nebo celého systému, nicméně v reálném světe toto často není možné.

Využijte cache mechanismus k tomu, aby jste dokázali docílit co nejnižšího průměrného času volání třídy XYZ, bez modifikaci třídy YYY. Viz konkrétní zadání dále.

Repozitář

https://github.com/xmorfeus/cacheexamle

Postup

1. nakopírovat složky tomcat8080 a tomcat8090
2. ve složce bin spustit ./startup.sh (funguje pro java 8)
3. otestovat že běží 127.0.0.1:8080, 127.0.0.1:8090

Zadání číslo 1: Optimalizace na nodu pomocí naivní implementace

curl 127.0.0.1:8090/cacheServlet/getstaticobject

Upravovat GetStaticObject/public void service

Změřit pomocí callingApp (připravená v gitu)

Výhody a nevýhody naivní implementace

1. Rychlá a jednoduchá implementace
2. Nemá skoro žádnou režii

3. Neřeší aktuálnost dat
4. Nelze použít na clustru

Zadání číslo 2: optimalizace na nodu pomocí implementace 3. strany

1. Upravte původní kód za pomoci JBOSS TreeCache
2. Dependency jsou přidané v projektu

Výhody a nevýhody referenční implementace

1. Hotové řešení
2. Zajišťuje kompletní práci s cache

Zadání číslo 3: zajistěte aktuálnost dat pomocí referenční implementace

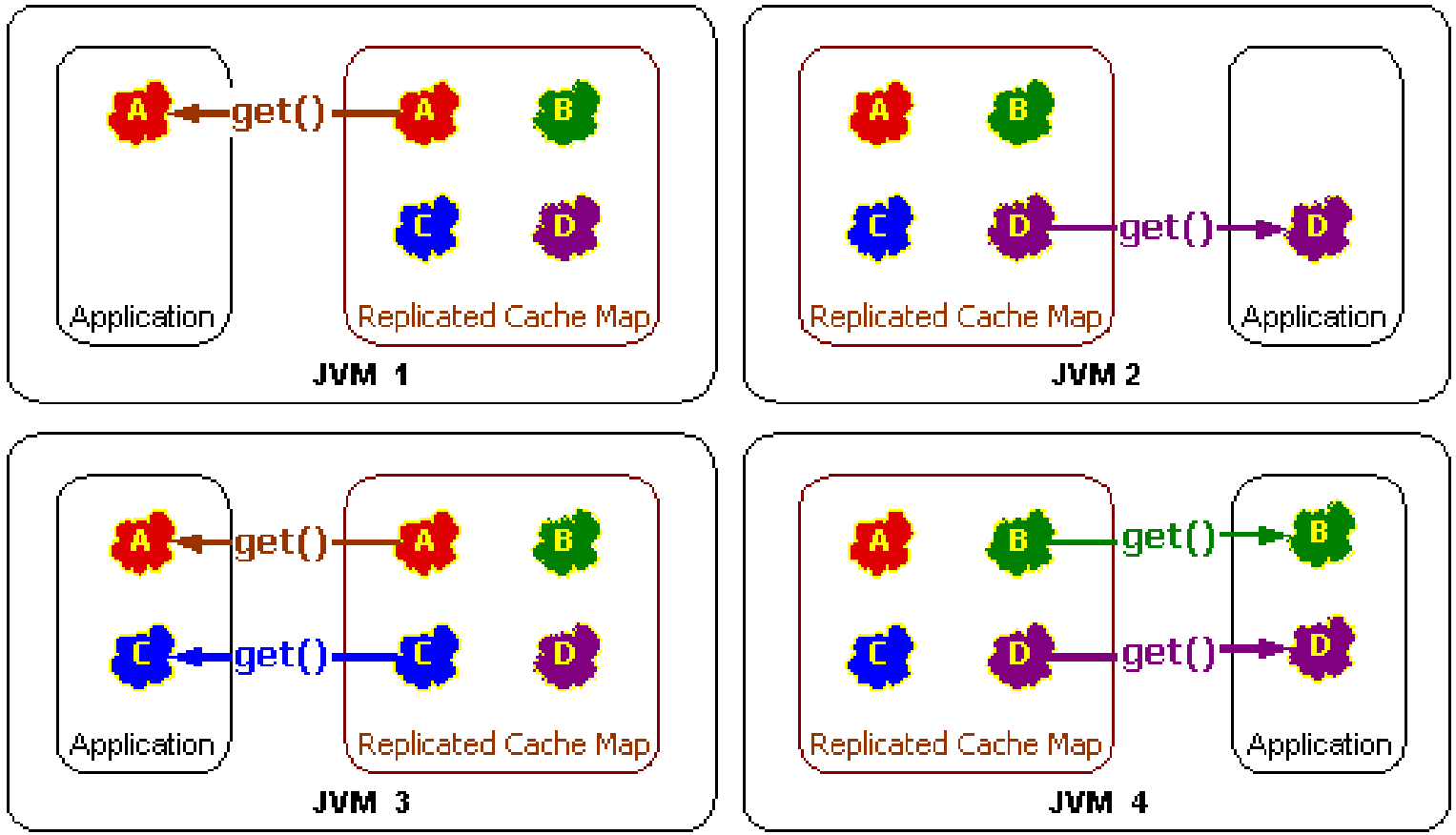
1. curl 127.0.0.1:8090/cacheServlet/getchangingobject
2. Upravovat GetChangingObject/public void service
3. Nastavte správnou konfiguraci
4. Změřit pomocí callingApp (připravená v gitu)

```
<config>
        <attribute name="wakeUpIntervalSeconds">10</attribute>
        <region name="/_default_">
           <attribute name="maxNodes">5000</attribute>
           <attribute name="timeToIdleSeconds">5</attribute>
           <attribute name="maxAgeSeconds">5</attribute>
        </region>
    </config>
```

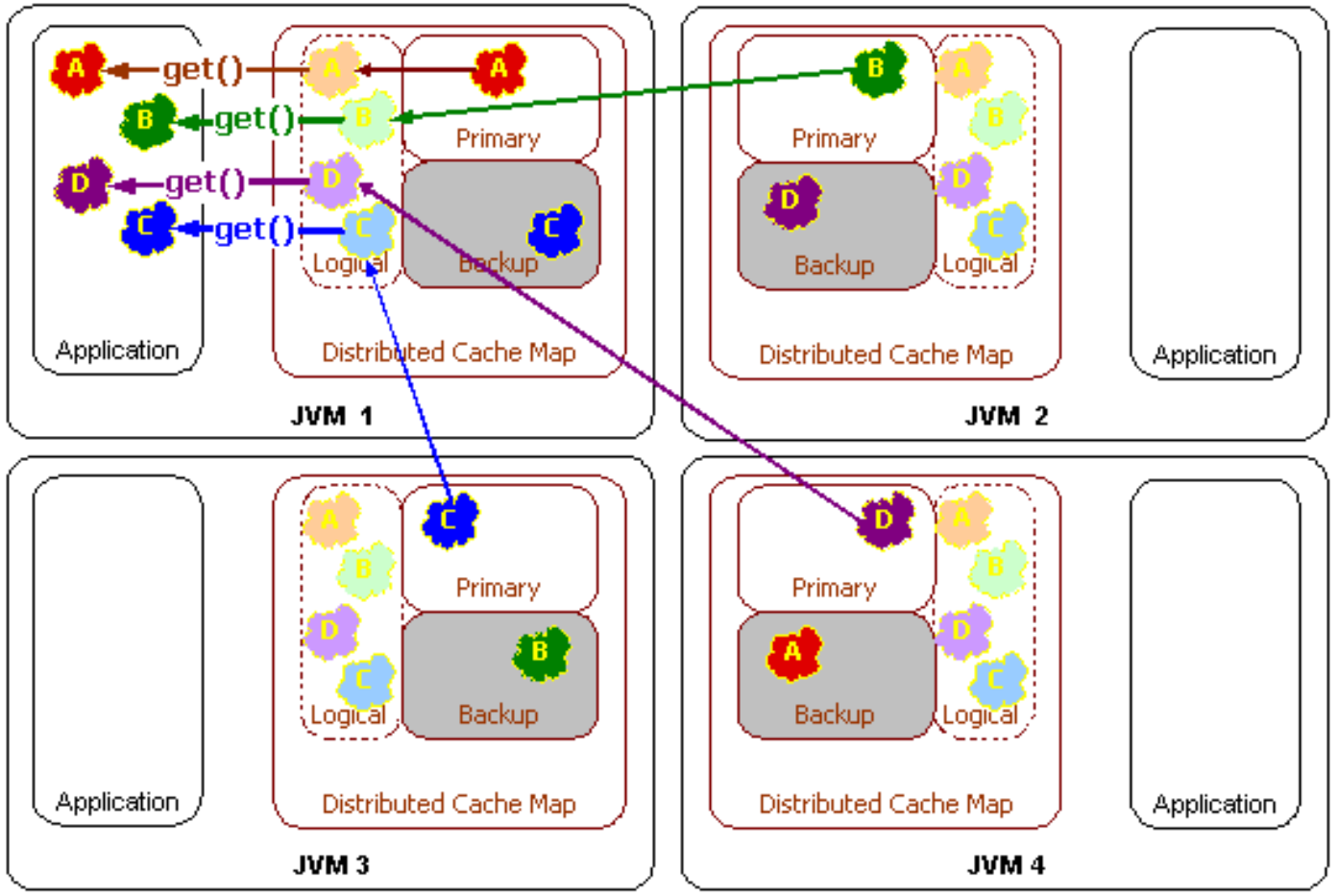
Clustred cache

1. Cluster = seskupení serverů, které spolupracují a na venek se tváří jako jeden
2. Izolovaná
    1. Každá aplikace má svou vlastní cache
    2. Cache spolu nekomunikují
    3. Použití pokud není
3. Replikovaná
    1. Všechny prvky jsou replikovány do všech ostatních cache
    2. Data jsou dostupná všude
4. Distribuovaná
    1. Každý server je zodpovědný za konkrétní část dat

Replikovaná cache



Distribuovaná cache



Zadání číslo 4: Zajistěte optimalizace na všech node pomocí cluster cache

1. Využijte Hazelcat ClusterCache
2. Dependency jsou přidané v projektu

Zadání číslo 5: Zajistěte optimalizace na všech node pomocí cluster cache a dbejte na aktuálnost dat

1. Doplňte configuraci Hazelcast ClusterCache
2. Nastavte správné hodnoty

```
<map name="default">
      <in-memory-format>OBJECT</in-memory-format>

      <time-to-live-seconds>20</time-to-live-seconds>
      <max-idle-seconds>20</max-idle-seconds>
      <eviction-policy>LRU</eviction-policy>
      <max-size policy="PER_NODE">5000</max-size>

   </map>
```

Výhody a nevýhody cluster cache

Data se získají pouze jednou a následně jsou dostupná na všech node.

Zneplatnění či reset cache má vliv na všechny nody v clusteru.

Snižuje zátěž prvku (Single Point of Failure – například DB).

V případě replikovaných cache, odpojení jednoho node z clusteru nevynutí přenačtení cache na všech serverech.

Nutná synchronizace dat.

Zápis zpravidla znamená zneplatnění cache.

Výkonový problém jednoho node v clusteru způsobí výkonový problém celé cache.

Problém přenosu velkého množství dat mezi lokalitami.

Problém při alokování výlučného zámku na cache.

Kvalitní řešení jsou často velmi drahá. Free verze neumožňují správnou diagnostiku a mnohdy cache musí nastavovat vývojář bez expertní znalosti dané implementace.

Restart serveru nevynutí reset cache.