Semestrální práce z předmětu KIV/OS

Distribuovaná bankovní aplikace

Zdeněk Valeš

26.11. 2019

1 Zadání

Implementuje distribuovanou bankovní aplikaci, která využívá sekvencer k řazení příchozích požadavků.

Popis chodu aplikace:

- všechny servery začínají se stejným zůstatkem na účtu 5000000.
- klient náhodně generuje částku v intervalu [10000,50000] a náhodně vybere operaci CREDIT či DEBIT a pošle operaci s částkou sekvenceru. Klient vygeneruje volitelný počet operací (např. číslo předané z příkazové řádky).
- sekvencer přiřadí jednoznačný identifikátor (rostoucí sekvence celých čísel) zprávě klienta a pošle zprávu shuffleru, který každou správu pošle na všechny bankovní servery (1 až N, kde N > 3) v náhodném pořadí.
- bankovní servery se musí vypořádat se zprávami, které chodí mimo původní pořadí, aby operace nad jejich lokálním účtem byly provedeny ve správném pořadí tak, jak je poslal klient.
- po ukončení činnost klienta a zpracování všech operací musí být v případě správné funkce bankovních serverů na všech stejný zůstatek.
- k vytvoření infrastruktury použijte nástroje Vagrant (+VirtualBox), k instalaci, konfiguraci a nasazení aplikace využijte jeho shell provisioning.
- aplikaci můžete implementovat v jazyce Java, nebo Python s využitím již existujícího software, který vám usnadní implementaci jednotlivých modulů.
- rozhraní serverů implementujte jako REST API. Popis všech rozhraní ve formátu OpenAPI bude součástí zdrojových kódů. Data přenášejte ve formátu JSON.

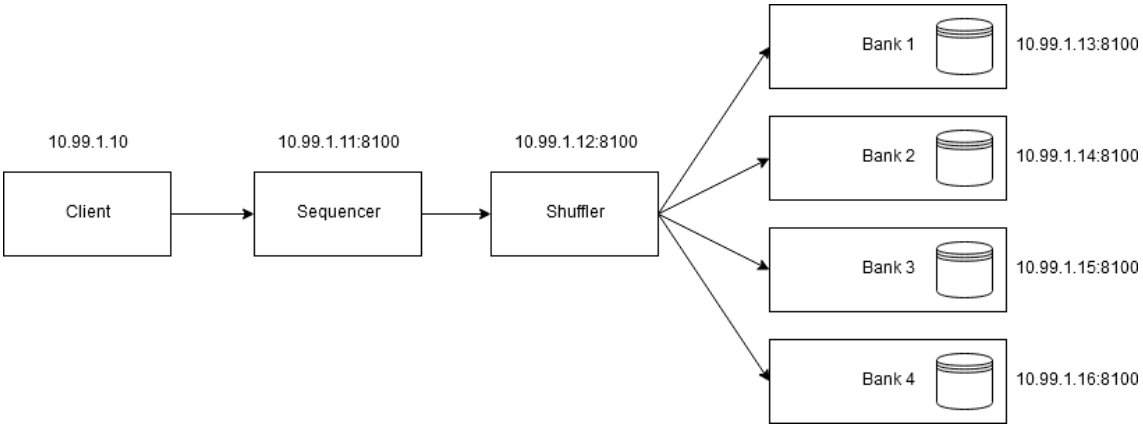
2 Popis implementace

K implementaci jsem si vybral jazyk Python (verze 3). Server, který realizuje REST API je postaven na knihovně Bottle, k volání API je použita knihovna requests. Systém obsahuje celkem 4 bankovní servery.

2.1 Topologie systému

Topologie systému, společně s IP adresou virtuálního stroje a portem, na kterém je přístupné REST API je znázorněna na obrázku **??**. Každý virtuální stroj je založen na image `ubuntu/trusty32` – můj počítač nepodporuje virtualizaci 64 bitových systémů.

Dokumentaci API jednotlivých serverů je přiložena v souborech `sequencer-api.yml`, `shuffler-api.yml` a `bank-api.yml`.

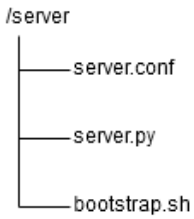


Obrázek 1: Topologie systému

2.2 Obecná struktura projektu

Zdrojové kódy projektu se nachází ve složce `src`. Každý server je umístěn ve vlastní složce (`client`, `bank-1`, ...). V adresáři `src` je také umístěn `Vagrantfile` obsahující konfiguraci nasazení virtuálních strojů.

Server je tvořen python skriptem, konfiguračním souborem pro `upstart` a skriptem `bootstrap.sh`, který server inicializuje a spouští. Každý server je na virtuálním stroji veden jako služba. Tím je možné daný server pohodlně vypínat/zapínat. Každý ze serverů loguje svou činnost do souboru `log.txt`, skrze který je možné za běhu kontrolovat stav systému.



Obrázek 2: Obecná struktura složky serveru

2.3 Client

Klient je ze všech modulů nejjednodušší, neobsahuje vlastní server a pouze generuje pokyny k transakcím. Počet pokynu, který má klient generovat a adresa sekvenceru jsou skriptu předány parametrem při spuštění.

2.4 Shuffler

Shuffler má interní frontu (o velikosti 10) do které ukládá pokyny k operacím přijaté od sekvenceru. Po naplnění je fronta zpřeházena a pokyny jsou postupně odeslány na

všechny bankovní servery.

Adresy bankovních serverů jsou načítány z konfiguračního JSON souboru, jehož cesta je shuffleru předána parametrem.

2.5 Bank N

Bankovní server přijímá pokyny k transakcím od shuffleru a ukládá je do interní haldy. Na vrchu této haldy je pokyn s nejmenším časovým razítkem (ta generuje sekvencer). Při spuštění bankovního serveru je nastaven interní čítač očekávaných id na hodnotu, která musí odpovídat prvnímu id generovanému sekvencerem. Pokud je na vrcholu haldy pokyn s $id = expected\_id$, je vyjmut a proveden. Po každé provedené transakci je do souboru `balance.txt` zapsán současný zůstatek na účtě.

Vzhledem k nutnosti 'synchronizace' prvního id generovaného sekvencerem a počátečního `expected_id` bankovního serveru je v případě restartu sekvenceru nutné restartovat také všechny bankovní servery (a naopak).

Každý bankovní server má vlastní databázi běžící na daném virtuálním stroji.

3 Spuštění

Systém je možné spustit příkazem `vagrant up` v adresáři `src`. Stav klienta je možné kontrolovat skrze ssh na vrituální stroj příkazem `status client`. V případě, že klient doběhl, jeho status bude `stop/waiting`.

Po doběhnutí klienta lze zkontrolovat zůstatky bankovních serverů. Zůstatek je na každém serveru uložen v souboru `/home/vagrant/bank/balance.txt`.

4 Závěr

Práci se mi podařilo úspěšně implementovat. Míchání požadavků shufflerem by bylo lepší realizovat skrze náhodné zpoždění, systém by tak více odrážel reálný stav. Zapisovat aktuální zůstatek na účte do souboru po každé provedené transakci je není obecně dobrá praktika, ale zde to usnadňuje testování a kontrolu výsledků.