Semestrální práce z předmětu KIV/DS

Globální stav v distribuovaném systému

Zdeněk Valeš

10.1. 2020

1 Zadání

Implementace ZeroMQ služby s integrovaným Chandy-Lmportovým algoritmem pro získání konzistentního snímku globálního stavu.

Popis chodu aplikace:

- Implementujte bankovní službu s operacemi (lze využít implementaci z předchozí úlohy) credit a debit tak, že obě operace lze provést na základě zprávy doručené do vstupní ZeroMQ fronty následovně: pokud klient chce poslat částku do jiné banky, odečte částku a pošle zprávu CREDIT s částkou do jiné banky (fronty). Pokud chce klient inkasovat částku z jiné banky, pošle pouze zprávu DEBIT s částkou, přičemž druhá strana částku odečte a pošle ji ve zprávě CREDIT.
- Všechny uzly jsou identické, kromě IP adresy a jména uzlu, jejich počet je v rozmezí 2-5
- Každý uzel po startu začíná se zůstatkem na účtu 5.000.000.
- Uzly náhodně vyberou jednu z operací CREDIT či DEBIT s náhodnou částkou v intervalu ¡10.000, 50.000¿ a odešlou zprávu s operací na náhodně vybraný sousední uzel. Pokud uzel na svém účtu nemá dostatečnou částku, operaci odmítne.
- Uzly spolu komunikují pouze přes vstupní ZeroMQ fronty (podle vzoru PAIR, viz, které představují komunikační kanály. Zprávy přenášejte v JSON formátu
- Topologii si zvolte sami, ale je nutné ji uvést v průvodní dokumentaci
- Jako servisní službu implementujte Chandy-Lamportův distribuovaný algoritmus pro získání konzistentního snímku globálního stavu podle stejného principu jako bankovní API (zpráva "marker" ve vstupní frontě – viz popis algoritmu)
- Spuštění Chandy-Lamportova algoritmu zahajte posláním zprávy MARKER do libovolné fronty (z příkazové řádky na libovolném uzlu).
- Ošetřete i případ, že může být v jednom okamžiku spuštěno více snapshotů
- Stavem uzlu se rozumí zůstatek na účtu v okamžiku přijetí prvního markeru, stavem kanálu seznam zpráv s bankovními operacemi přijatými po prvním markeru (viz popis algoritmu)
- Jakmile detekujete ukončení algoritmu, všechny uzly pošlou svoje uložené stavy do vyhrazené fronty, ze které je vyhrazený proces vybírá a zobrazuje (stačí na konzoli)
- K vytvoření infrastruktury použijte nástroje Vagrant a VirtualBox
- Aplikaci můžete implementovat v Jazyce Java nebo Python s využitím již existujícího software, který vám usnadní implementaci jednotlivých modulů a jejich vzájemnou komunikaci.

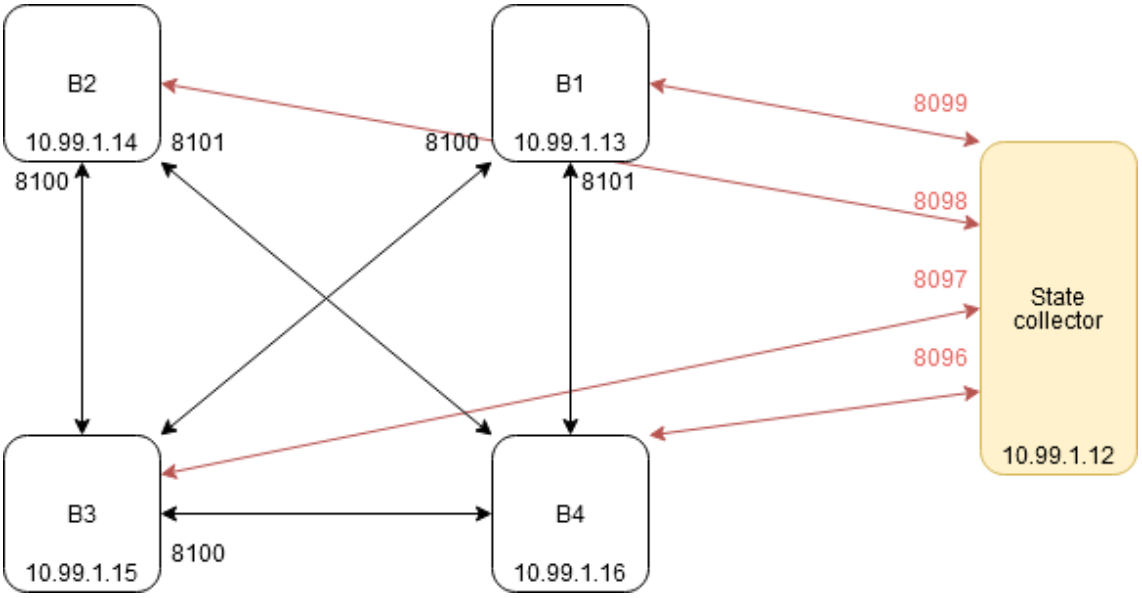
2 Popis implementace

K implementaci jsem si vybral jazyk Python (verze 3). Banky a služba pro sběr globálního stavu spolu navzájem komunikují skrze ZroMQ fronty. Systém obsahuje 4 bankovní servery a jednu službu pro sběr stavu.

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 Konfigurace

TODO

2.3 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

2.4 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.

2.4.1 Chandy-Lamport

TODO

2.5 Služba pro sběr stavů systému

TODO

3 Spuštění

TODO 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

TODO 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.