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:

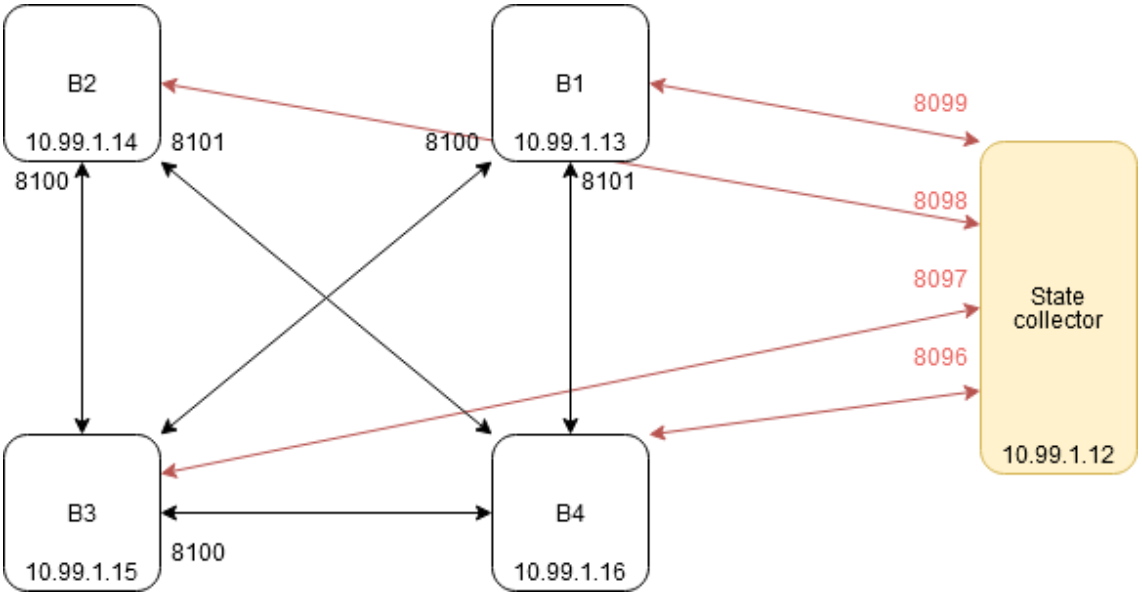
- Implementujte bankovní službu s operacemi (lze využít implementaci z předchozí úlohy) credit a debit tak, že obě operace lze provést na základě zprávy doručené do vstupní ZeroMQ fronty následovně: pokud klient chce poslat částku do jiné banky, odečte částku a pošle zprávu CREDIT s částkou do jiné banky (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 adresami virtuálních strojů a porty, na kterých služby poslouchají je znázorněna na obrázku 1. Každý virtuální stroj je založen na image `ubuntu/trusty32` – můj počítač nepodporuje virtualizaci 64 bitových systémů.



Obrázek 1: Topologie systému

2.2 Konfigurace

Bankovní služba při spuštění čte z konfiguračních souborů porty, na kterých má poslouchat a adresy, na které se má připojit. Adresy pro připojení k ostatním bankám a state collectoru jsou ve zvláštních souborech. Příklad csv souboru pro konfiguraci propojení bank:

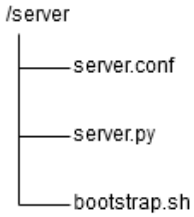
```
1,8100,8101
2,8100,8101
3,8100
4
3,10.99.1.13:8100,10.99.1.14:8100
4,10.99.1.13:8101,10.99.1.14:8101,10.99.1.15:8100
```

První sloupec je ID banky, v následujících sloupcích jsou buď porty na kterých banka poslouchá, nebo adresy, kam se připojí.

2.3 Obecná struktura projektu

Zdrojové kódy projektu se nachází ve složce `src`. Bankovní služba je umístěna ve složce `bank`, služba pro sběr globálního stavu je umístěna ve složce `state-collector`. V adresáři `src` je také umístěn `Vagrantfile` obsahující konfiguraci nasazení virtuálních strojů.

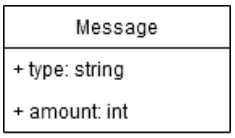
Server je tvořen python skriptem, konfiguračním souborem pro `upstart` a skriptem `bootstrap.sh`, který server inicializuje a spouští. Každý server je na virtuálním stroji veden jako služba. Tím je možné daný server pohodlně vypínat/zapínat. Každý ze serverů loguje svou činnost do souboru `log.txt`,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



Obrázek 2: Obecná struktura složky serveru

2.4 Bank N

Každá banka má své unikátní ID. Po spuštění bankovní server přijímá a náhodně generuje zprávy, které posílá svým sousedům. Zprávy jsou odesílány pouze na aktivní sockety (je připojen protějšek). Každý bankovní server má vlastní databázi běžící na daném virtuálním stroji. Struktura zprávy posílané mezi bankami je znázorněna na obrázku 3 – `type` je `CREDIT`, nebo `DEBIT` a `amount` je částka, se kterou má být daná operace provedena. Speciálním typem je pak zpráva `REFUSE` s částkou nastavenou na -1, která oznamuje neplatnou operaci (na účtu banky není dostatek peněz).

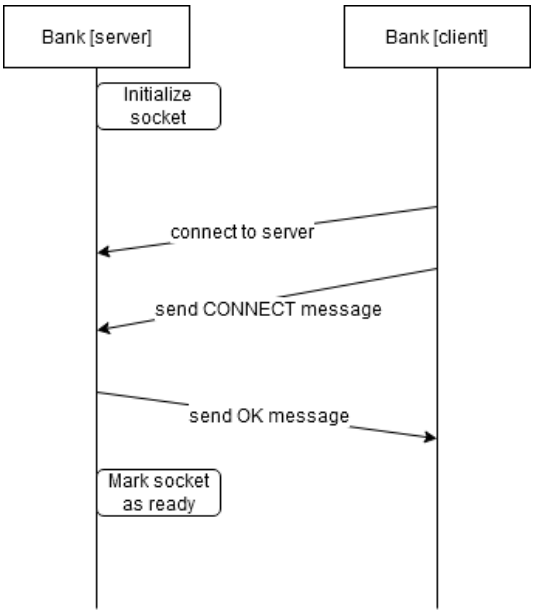


Obrázek 3: Struktura zprávy posílané mezi bankami

2.4.1 Připojení k bankovní službě

Bankovní služby při připojení provádějí zjednodušený handshake, aby server (banka, která na socketu poslouchá) mohl rozpoznat, kdy se protějšek připojil. Handshake (znázorněn

na obrázku 4) spočívá v odeslání `CONNECTION` zprávy ihned po připojení se k socketu. Po přijetí a potvrzení `CONNECTION` zprávy si server označí daný socket jako připravený a začne na něj odesílat zprávy.



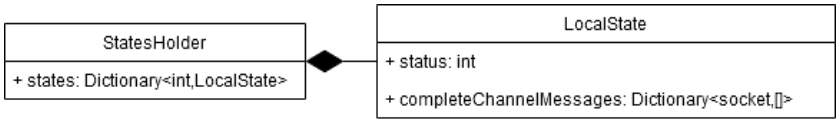
Obrázek 4: Handshake mezi bankovními službami

2.4.2 Chandy-Lamport

Součástí smyčky přijmi zprávy/odešli zprávy bankovního server je také kontrola, zda se má spustit algoritmus pro detekci globálního stavu. Server pravidelně kontroluje přítomnost souboru pojmenovaného `MARKER` ve svém pracovním adresáři. Pokud tento soubor existuje, server spustí Chandy-Lamportův algoritmus a rozešle zprávy `MARKER` svým sousedům.

Po dokončení algoritmu odešle server svůj zaznamenaný stav a stav komunikačních kanálu state collectoru, který jej zaloguje.

Aby bylo možné spouštět více snapshotů najednou, obsahuje každá `MARKER` zpráva také unikátní ID, na jehož základě je stav zaznamenán. Jako ID je použito ID banky, předpokládá se, že jeden uzel nemůže algoritmus spustit vícekrát najednou (musí tedy počkat až jím iniciovaný algoritmus doběhne). Struktura, která v každé bance udržuje lokální stavy (1 stav pro 1 marker id) je znázorněna na obrázku 5. Výsledný globální stav je pak složen z jednotlivých lokálních stavů všech bank.



Obrázek 5: Struktura pro udržení lokálních stavů

2.5 Služba pro sběr stavů systému

State collector po spuštění poslouchá na předem konfigurovaných portech (na které se připojují banky) a vypisuje příchozí zprávy so souboru `log.txt` umístěného v pracovním adresáři.

Výpis stavové zprávy má následující formát:

```
Status message: marker_id=%s; bank_id=%s; status=%s; channel_messages=%s
```

Kde `marker_id` je unikátní id použité k rozeznání jednotlivých `MARKER` zpráv. `bank_id` je id banky, která danou stavovou zprávu odeslala, `status` je zůstatek na účtu banky a `channel_messages` jsou zprávy přijaté bankou od odeslání `MARKER` do konce algoritmu.

3 Spuštění

Systém je možné spustit příkazem `vagrant up` v adresáři `src`. Sběr globálního stavu je možné spustit příkazem `touch MARKER` v adresáři `bank` kterékoliv bankovní služby. Globální stav je pak možné přečíst v souboru `state-collector/log.txt` (například příkazem `touch -f state-collector/log.txt`). Příklad výpisu logu, který obsahuje globální stav je na obrázku 6.

```
14:27:53,708 root INFO Status message: marker_id=4; bank_id=1; status=3742496; c
hannel_messages={};
14:27:53,731 root INFO Status message: marker_id=4; bank_id=3; status=7562544; c
hannel_messages={};
14:27:53,733 root INFO Status message: marker_id=4; bank_id=4; status=3727211; c
hannel_messages={};
14:27:53,738 root INFO Status message: marker_id=4; bank_id=2; status=4967749; c
hannel_messages={};
```

Obrázek 6: Příklad výpisu globálního stavu

Poznámka: všechny virtuální stroje mají v konfiguraci vypnutou HW virtualizaci (`["--hwvirtex", "off"]`). V případě, že by to při spuštění dělalo problémy, je možné ji zapnout.

4 Závěr

Práci se mi podařilo úspěšně implementovat. Bohužel, algoritmus je moc rychlý a nepodařilo se mi při testování zachytit stav tak, že by jeho součástí byly neprázdné stavy kanálů.