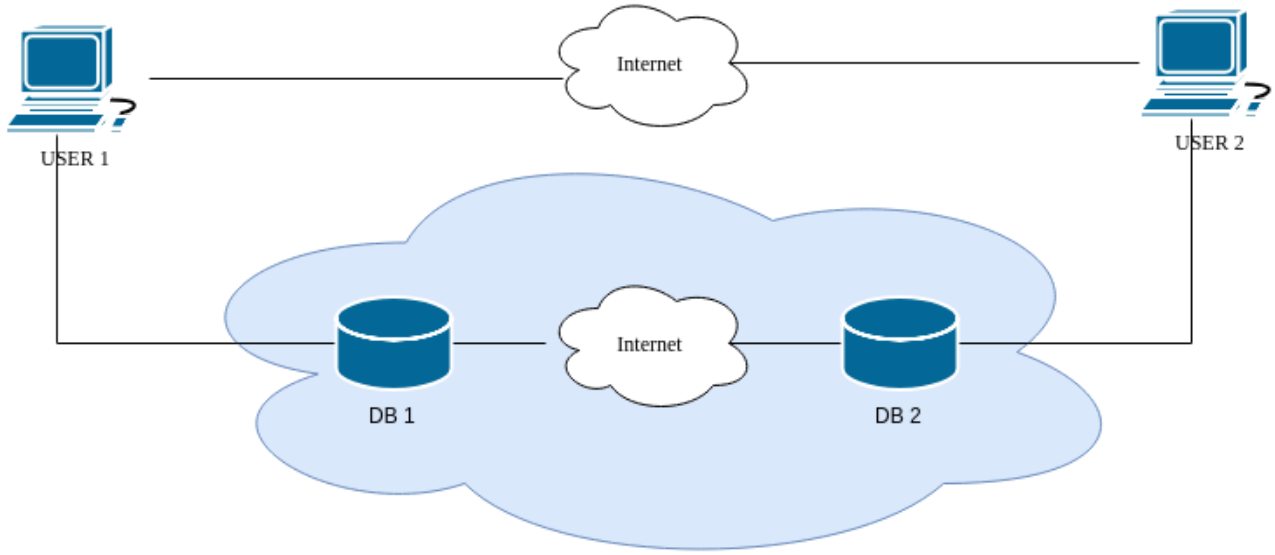
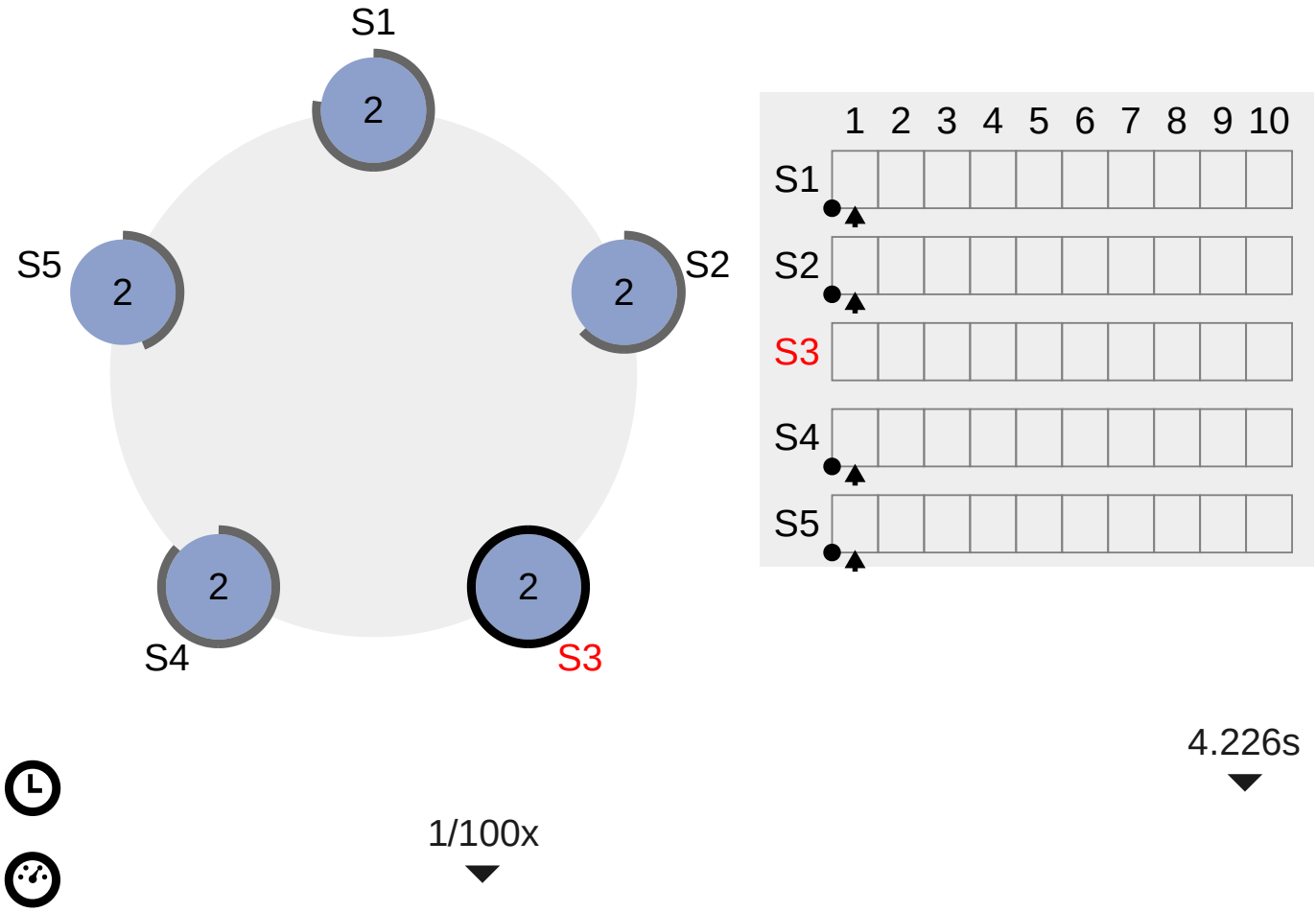
2. semestrální práce - Konsenzus v distribuovaném systému

Jedním ze základních požadavků na distribuovaný systém je schopnost dosahovat spolehlivých výsledků výpočtu i za předpokladu, že dochází ke zpomalování či úplné ztrátě zpráv, nebo i vypadávání celých procesů. V mnoha případech je podmínkou pro to, aby byl tento požadavek splněný, možnost shodnout se na hodnotě některé z proměnných, která je během výpočtu použita. Této shodě se v distribuovaných systémech říka **konsenzus**. Mezi reálné aplikace konsenzu patří například synchronizace hodin, výpočet PageRanku používaný Googlem, kontrola shluků mobilních robotů, vyvažování zátěže nebo **replikace dat v distribuované databáze**. A právě poslední zmíněný příklad bude úlohou, kterou budete v rámci semestrální práce řešit.



[/wiki/_detail/courses/b4b36pdv/tutorials/pdv_cloud.png?id=courses%3Ab4b36pdv%3Atutorials%3Asem2]

Algoritmus, který budete pro tento účel využívat, se nazývá **Raft** a byl probírán na přednášce. Jeho jednotlivé části pak byly odvozeny detailněji na cvičeních. Raft je příkladem algoritmu k dosažení konsenzu, který se stará o správu a replikaci logu, což z něj dělá dobrý stavební prvek pro distribuovanou databázi. Připomínáme, že kompletní pseudokód a vysvětlení algoritmu můžete najít zde [https://www.cs.princeton.edu/courses/archive/fall16/cos418/papers/raft.pdf]. K lepšímu pochopení algoritmu můžou posloužit zdroje k nalezení zde [https://raft.github.io]. Z posledního zdroje by se Vám mohla hodit i tato pěkná visualizace fungování Raftu:



Distribuvaný systém, se kterým budete pracovat, se skládá z dvou typů procesů: **klientský** proces a **serverový** proces. Procesů obou typů může být v systému mnoho. Můžete počítat s tím, že spojení mezi klienty a servery jsou spolehlivé a zprávy se na nich neztrácí, ani nedochází k jejich zpoždění. Klientské procesy jsou také spolehlivé a nevypadávají. Hlavním úkolem každého serverového procesu je udržovat svou vlastní kopii databáze (logu), konzistentní s databázemi ostatních serverových procesů. Serverový proces musí mít přehled o tom, který serverový proces je aktuálním lídrem, aby tomu mohl přizpůsobit své chování (stav). Za každých okolností, když potvrdíte klientovi nějaký požadavek jako zpracovaný (committed), je skutečně zpracovaný a tak i zůstane. Nemůže se stát, že byste klientovi řekli, že jste uložili nějáka data a ta by se pak ztratila.

V našem případě je databáze reprezentována jako zobrazení z klíče - K, reprezentovaného jako `String`, na hodnotu - H, reprezentovanou rovněž jako `String`, jedná se tedy o velice jednoduchou key/value databázi, která je v našem případě reprezentována rozhraním Map<String, String>. Klient nad touto databází může vykonávat 3 operace - `GET`, `APPEND`, `PUT`, které jsou popsány níže.

Klientské procesy mohou mít na kterýkoli ze serverových procesů následující požadavky:

- *Kde je aktuálním leaderem?* se klient ptá pomocí zprávy `ClientRequestWhoIsLeader`, a

- *Proveď operaci nad databází* klient vyžaduje zprávou `ClientRequestWithContent`. V této zprávě klient definuje na jakém klíči má operace proběhnout, s jakou hodnotou a o jakou operaci se jedná. Dvojici klíč KZ a hodnota HZ vrací funkce `getContent()` pro tuto zprávu. Typ operace lze zjistit pomocí volání funkce `getOperation()` a jde vždy o jeden z následujících:
  - `PUT` asociuje v databázi hodnotou HZ s klíčem KZ. Pod klíčem K=KZ bude tedy hodnota H=HZ.
  - `APPEND` funguje jako `PUT` v případě, že klíč K=KZ v databázi není. Pokud ale K existuje a má hodnotu H', potom dojde ke zřetězení původní hodnoty H' s hodnotou HZ.
  - `GET` se ptá na hodnotu klíče K=KZ v databázi. Hodnota HZ ve zprávě je v tomto případě vždy `null`.

Každý klientský požadavek má vlastní ID, reprezentované jako String. Toto ID lze zjistit voláním `getRequestId()` na přijaté zprávě. Je důležité, abyste při odpověďi na klientský požadavek, použili jako ID odpovědi identifikátor požadavku, jinak nebude klient schopný odlišit, na který požadavek mu došla odpověď.

Stáhněte si balíček sem_02.zip [https://cw.fel.cvut.cz/wiki/_media/courses/b4b36pdv/tutorials/sem_02.zip]. Vaším hlavním úkolem bude naimplementovat serverový proces, jehož hlavní třídou je třída `ClusterProcess`. Tato třída je již v balíčku předimplementována. Jako u předchozích domácích úloh z distribuovaných výpočtů i zde platí, že když se proces probudí, může zpracovat přijaté zprávy v metodě `act`. Serverové procesy musí být schopné reagovat na požadavky klientských procesů. Konkrétně musí:

- informovat klienta o tom, který serverový proces se domnívají že je leaderem, pomocí zprávy `ServerResponseLeader`. Konstruktor této zprávy vyžaduje ID požadavku (viz výše) a ID serverového procesu, který by měl být leaderem.
- informovat o úspěšnosti provedení operace. V případě, že klient žádá provést `PUT` nebo `APPEND`, odpovídá leader po potvrzení zpracování (commitu) zprávou `ServerResponseConfirm`. Pokud se klient ptá na hodnotu klíče pomocí `GET`, leader odpoví zprávou `ServerResponseWithContent`, do které vloží v konstruktoru hodnotu, kterou má pod klíčem uloženou. Pokud dotazovaný proces není leaderem, měl by o tom dát klientskému procesu vědět pomocí zprávy `ServerResponseLeader`.

Mimo třídu `ClusterProcess` můžete implementovat i řadu dalších tříd. Jistě budete potřebovat třídy pro jednotlivé typy zpráv v Raftu, hodit se mohou také třídy starající se o logy jednotlivých procesů. Jako u každé programovací úlohy, je dobré si Váš návrh tříd nejdříve pořádně rozmyslet a až poté začít implementovat. Tím můžete předejít případným budoucím problémům s laděním celého algoritmu.

Vaší implementaci můžete testovat pomocí scénářů definovaných ve třídě `RaftRun` v balíčku `evaluation`. Ve třídě `Scenario` v témže balíčku najdete popis parametrů distribuovaného prostředí. Doporučujeme si zkusit s parametry zahýbat a analyzovat, jak si Váš algoritmus se vznikajícími problémy poradí. My Váš kód budeme testovat na scénářích s různými nastaveními těchto prametrů.

Vyčleňte si pro implementaci Raftu dostatek času a začněte brzy! Na předchozích dvou domácích úlohách si již mnozí z Vás vyzkoušeli, že odladit i koncepčně jednoduchý algoritmus může být v distribuovaném prostředí složité. Pokud začnete dřív, budete se moci s problémy jednak lépe vypořádat, a druhak je i konzultovat se cvičícím.

Hodnocení úlohy

Za úlohu můžete získat maximálně **14 bodů** v závislosti na kvalitě a schopnostech Vašeho řešení. Body se Vám budou nasčítávat podle následujících podúloh.

1. *Volba leadera* **[3b]**. Za úspěšné zvolení leadera, o kterém ví všechny serverové procesy, dostanete 1b. Pokud je Váš algoritmus schopný leadera zvolit i po selhání několika procesů, dostanete další 1b. Poslední 1b získáte, pokud serverové procesy nového leadera zvolí správně i po obnovení několika dříve spadlých procesů.
2. *Replikace logu* **[8b]**. V tomto případě budete pracovat s odlehčenou verzí databáze, kdy budete zpracovávat jen požadavky `APPEND` se stále stejným klíčem. Budeme testovat kombinace tří parametrů sítě, kdy každý parametr může nabývat dvou hodnot. Za správně zpracovanou většinu požadavků a dobře zreplikované logy získáte v každém scénáři 1b, dohromady tedy 8b.
    1. První parametr říká, kolik klientů přistupuje současně k distribuované databázi: jeden či několik.
    2. Druhý parametr určuje, zda dochází k výpadkům a obnovování serverových procesů.
    3. Poslední parametr nastavuje spolehlivost zpráv: zda dojde k přerušení kontaktu mezi částmi sítě.
    4. Navíc musíte být schopni správně zpracovat většinu (někdy všechny) požadavky klientů. Vždy, když klientovi potvrdíte zpracování požadavku, požadavek musí být skutečně zpracovaný a tak i zůstat. Budeme kontrolovat stav a správnost logu.
3. *Plně funkční distribuovaná databáze* **[3b]**. Bude docházet ke stejným problémům jako u replikace logu. Navíc Vaše řešení musí zvládat zpracovat již všechny tři operace. Zatímco u druhého scénáře se mohly některé vyřízené požadavky opakovat, zde k tomu bude docházet zcela záměrně.

Primárním požadavkem na Vaše řešení je, aby bylo **distribuované** a bylo založené na **Raftu**! Pokud Vám BRUTE přidělí body, ale Vaše řešení nebude distribuované (procesy spolu musí komunikovat jen prostřednictvím zpráv, ve kterých nesdílejí přímo svoji lokální paměť) nebo nebude postavené na Raftu, znamená to nesplnění požadavků semestrální úlohy, a tedy i nulový počet bodů.
Všechny zazipované soubory z balíčku `student` odevzdávejte do systému BRUTE do pondělí **2. 7. 23:59 CET**.

courses/b4b36pdv/tutorials/sem2.txt · Last modified: 2023/06/06 16:00 by macejp1