Hirschberg-Sinclair Algorithm

Author: Dmytro Rastvorov

Semestral project
GitHub: github.com/UnknownPug/Hirschberg-Sinclair-algorithm
Subject: B2M32DSVA
Study program: Software Engineering and Technology

Contents

Základní funkce algoritmu:

Co je to za algoritmus

Hirschbergův-Sinclairův algoritmus je distribuovaný algoritmus určený k řešení problému volby vůdce v synchronní kruhové síti. V mé implementaci byly používán taky Leader Election algoritmus. Pro napsání aplikace byl použít jazyk Kotlin RMI.

Co si uzly pamatují?

Každý uzel bude obsahovat následující údaje:

1) Id
2) Name
3) IP Address
4) Port
5) IP Address sousedního vlákna
6) Sousední optický port
7) Host a port sousedních vláken (right, nNext, left)
8) Host a port aktuálního vedoucího

Jak jsou uzly propojeny

Uzly jsou propojeny tak, že je spuštěn vůdce, který je staticky vybrán (vlákno 2010). V argumentech tohoto vlákna je uveden pouze jeho název, id, ip adresa a port. Sousední vlákno bude mít kromě již zmíněného jména, id, ip adresy a portu také ip adresu a port svého souseda, tj. dříve volaného vlákna. Vytvoříme tedy řetězec, který bude vypadat takto: 2010 → 2020 → 2030 → 2040 → 2050 → 2060
Pokud se pokusíte spustit řetězec s vláknem 2030, tak se to nepovede, protože nezná svého předchozího souseda.

Proces obnovy topologie v tomto systému je popsán následovně:

1. Soused, který udržuje přímé spojení s nejbližším vláknem, odešle zprávu o ztrátě uzlu.
2. V reakci na přijatou zprávu se aktivuje algoritmus obnovy topologie.

3. Program zavolá metodu odpovědnou za zpracování ztracených vláken, která nebyla nalezena v řetězci. Zde se zkontroluje, zda se ztracené vlákno rovná svému sousedovi.
4. Pokud se vlákno shoduje se svým sousedem, jsou přenášena aktuální data, což vede k přerušení spojení se ztraceným uzlem.
5. Poté následuje kontrola, zda ztracená adresa byla vedoucí. Pokud se to potvrdí, systém informuje o úmrtí vedoucího a zahájí proces volby nového vedoucího.
6. Pokud adresa není adresou lídra, neprovedou se žádné další akce.
7. Pokud vlákno neodpovídá svému sousedovi, zavolá se metoda odpovědná za zpracování ztracených vláken sousedního uzlu.

Kdy a jak se provádí implementovaný algoritmus

Při spuštění každého uzlu zvlášť se spustí řetězec vláken. Poté máme automaticky vybraného lídra, který byl staticky vybrán v softwaru, tedy vlákno 2010. Jakmile vlákno 2010 odumře, řetězec bude moci pokračovat v odesílání zpráv, dokud nebudou ovlivněna sousední vlákna odumřelého vlákna. Jakmile je jedno z těchto vláken zasaženo, spustí se algoritmus:

1) Nejprve se inicializují nResp, který odpovídá za odpovědi přijaté uzly, a respOK, který odpovídá za to, zda byly odpovědi přijaty včas.

2) Spuštění volební metody nastaví její stav kandidáta a hloubku prohledávání lmax na 1, a dokud má stav kandidáta, vynuluje počet odpovědí. Po odeslání kandidáta svým sousedům (tj. pravému a levému vláknu) a čekání na jejich odpověď nebo čekání na hodnotu vítěze rovnou našemu vláknu. Jakmile program obdrží jednu z těchto odpovědí, synchronně zkontroluje, zda respOK nenabyl hodnoty false. Pokud přijala false, pak se jeho stav označí jako ztracený.

3) Poté je zahájena kandidatura.

- Pokud je id lídra menší než id vlákna, které jsme vybrali, odešle se na adresu odesílatele odpověď se stavem false, což znamená, že id již není kandidátem, a v případě, že se ještě nezúčastnil, začne hlasovat, takže se pokusí stát kandidátem.

- Pokud se ukáže, že id lídra je větší než id našeho vybraného vlákna, oznámí, že naše vybrané vlákno ztratilo, minimální hloubka se zvýší, zkontroluje se, zda je minimální hloubka menší než maximální hloubka. Pokud ano, odešle zprávu kandidáta na odpovídající uzel během procesu volby. Samotná zpráva je předána uzlu vrácenému metodou getProxyToPassMessage. Metoda getProxyToPassMessage vrací RMI proxy uzlu, kterému má být zpráva odeslána. Pokud je odesílatelem pravý soused, je vrácena proxy levého souseda a naopak. V opačném případě s minimální a maximální hloubkou bude na adresu odesílatele odeslána odpověď se stavem true. Odpoví tedy tomu, kdo zprávu odeslal (tomuto kandidátovi), s tím, že do této hloubky je nejlepším kandidátem.
- Pokud se ID shodují, znamená to, že zpráva prošla kolem a neexistuje žádný nejlepší kandidát. V případě, že ještě nemá status ELECTED, získá status ELECTED, nastaví také ID vítěze a odešle zprávu do celého kola.

4) Volání metody odpovědi
Zkontrolujeme, zda se id rovná id skutečného vlákna. Pokud ano, pak zvýšíme nResp o 1 a také nastavíme respOK, čímž řekneme, zda je respOK skutečně pravdivý a odpověď na response je skutečně pravdivá.

5) Volání zvolené metody
Nejprve zkontrolujeme, zda se id vůdce nerovná id našeho aktuálního vlákna. Pokud ano, spustíme getProxyToPassMessage, kterému jako parametr nastavíme adresu odesílatele, a z něj zavoláme metodu elected, které předáme id vítěze, jeho adresu a adresu aktuálního vlákna. Poté nastavíme aktuálnímu vůdci adresu vítěze, jeho id a stav NOT_INVOLVED.

Práce s uzly (Odstranění a přidání 5 uzlů)

Protože mám mezi jednotlivými vlákny závislost na kruhu, odstraňoval jsem jedno po druhém, dokud nezůstalo jen jedno, a kontroloval jsem, zda topologie funguje a vlákna přijímají informace. Pak jsem začal jedno po druhém přidávat zpět a kontroloval, zda mají aktuální informace a mohou posílat zprávy svým sousedům. Vše fungovalo bez problémů.

Spuštění uzlů najednou

Začal jsem tím, že jsem na každém virtuálním počítači spustil 6 vláken, spustil projekt přes terminál a zkusil vzájemně pingnout, abych zjistil, zda vše funguje. Po pingu jsem nastavil funkci readln(), aby blokoval výběr vůdce. Vlákna pak čekala na mé rozhodnutí a po stisknutí klávesy "Enter", která odemkla výběr lídra, jsem získal lídra vlákna.

Závěr

V tomto předmětu jsem byl schopen se naučit Hirschbergův-Sinclairův algoritmus, porozumět jeho struktuře a fungování. Mohl jsem si také vyzkoušet jazyk Kotlin, díky čemuž jsem pochopil jeho syntaxi, a také jsem se dozvěděl, jak funguje rozhraní RMI. Jsem vděčný vyučujícímu Ing. Petru Macejko za vklad do našich znalostí!