VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

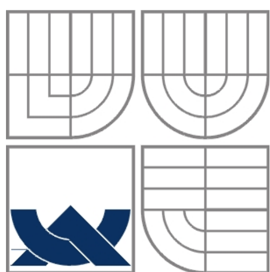
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

PARALELNÍ A DISTRIBUOVANÉ ZPRACOVÁNÍ ROZSÁHLÝCH TEXTOVÝCH DAT

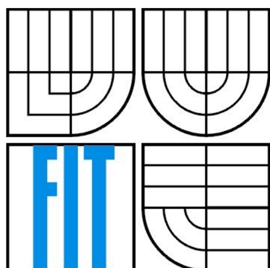
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE IVAN STRAKA
AUTHOR

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDI

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

PARALELNÍ A DISTRIBUOVANÉ ZPRACOVÁNÍ ROZSÁHLÝCH TEXTOVÝCH DAT

PARALLEL AND DISTRIBUTED PROCESSING OF LARGE TEXTUAL DATA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE IVAN STRAKA
AUTHOR

VEDOUCÍ PRÁCE doc. RNDr. PAVEL SMRŽ, Ph.D.
SUPERVISOR

BRNO 2015

Abstrakt

Tato práce se zabývá distribuovanými systémy, problémy spojenými s nimi, distribucí úloh a vyvažováním zátěže. Popisuje návrh a implementaci distribuovaného systému pro zpracování rozsáhlych textových dat, jeho architekturu, vyvažování zátěže, paralelní zpracování rozsáhlych textových dat, komunikaci mezi uzly, detekci chyb a zachování konzistence.

Abstract

This thesis deals with distributed systems, problems related to them, distribution of computing power and load balancing. It describes design and implementation of the distributed system for processing of large textual data, its architecture, loadbalancing, parallel processing of large textual data, communication between nodes, fault detection in communication and maintaining consistency.

Klíčová slova

distribuovaný systém, distribuce úloh, zpracování textových dat

Keywords

distributed system, task distribution, processing of textual data

Citace

Ivan Straka: Paralelné a distribuované spracovanie rozsiahlych textových dát, bakalárska práca, Brno, FIT VUT v Brně, 2015

Paralelní a distribuované zpracování rozsáhlých textových dat

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením doc. RNDr. Pavla Smrže, Ph.D.
Další informace mi poskytl Ing. Jan Kouřil.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……………………
Ivan Straka
20.5.2015

Poděkování

Rád by som vyjadril nesmiernu vďaku za rady, priateľské konzultácie, trpezlivosť a pripomienky vedúcemu práce pánovi *doc. RNDr. Pavlovi Smržovi, Ph.D. Ďalej by som rád poďakoval pánovu Ing. Janovi Kouřilovi za technické rady a pomoc pri vykonávaní experimentov.*

© Ivan Straka, 2015
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

Obsah....1
Zoznam tabuliek ilustrácií....3
1 Úvod....4
2 Distribuovaný systém....5
2.1 Vlastnosti distribuovaného systému....5
2.1.1 Škálovateľnosť....5
2.1.2 Výkon....6
2.1.3 Priepustnosť....6
2.1.4 Dostupnosť....6
2.1.5 Odolnosť proti chybám....6
2.2 Model distribuovaného systému....6
2.3 Komunikácia....7
2.3.1 Blokovaná a neblokovaná komunikácia....7
2.3.2 Spoľahlivosť komunikácie....9
2.3.3 Architektúra siete....10
2.4 Konsenzus....11
2.5 Architektúra distribuovaného systému....11
2.5.1 Centralizované riadenie....11
2.5.2 Decentralizované riadenie....12
2.6 Čas....13
2.6.1 Lamportov logický čas....13
2.6.2 Vektorové hodiny....14
2.7 Vzájomné vylúčenie....15
2.7.1 Centralizovaný algoritmus....15
2.7.2 Decentralizovaný algoritmus....16
2.7.3 Ricartov Agrawalov distribuovaný algoritmus....16
2.8 Voľba koordinátora....17
2.9 Distribuovaná zdielaná pamäť....17
2.9.1 Replikácie....18
2.9.2 Model konzistencie....18
2.10 CAP teorém....19
2.11 Detektor zlyhania....19

2.11.1 Vlastnosti detektorov zlyhaní....20
2.11.2 Typy detektorov zlyhaní....21
3 Distribúcia výpočtu....21
3.1 Konkurentné modely....22
3.1.1 Paralelní robotníci....22
3.1.2 Zreťazení robotníci....22
3.1.3 Paralelne zreťazení robotníci....23
3.2 MapReduce....23
4 Návrh a popis implementácie....24
4.1 Ciele systému....24
4.2 Model....25
4.2.1 Koncept....25
4.2.2 Popis implementácie....28
4.3 Detektor zlyhania....37
4.4 Spracovanie súborov na jednom výpočtovom uzle....37
4.4.1 Návrh....37
4.4.2 Popis implementácie....39
4.5 Návrh a spôsob implementácie prenosu správ v sieti....40
4.6 Konfigurácia....41
4.6.1 Forma konfiguračného súboru prepínača -g....42
4.6.2 Forma konfiguračného súboru prepínača -c....42
5 Experimenty....44
5.1 Spracovanie dataset CommonCrawl....44
5.2 Spracovanie súborov na výkonnejších strojoch....45
5.3 Porovnanie so systémom Hadoop....46
5.3.1 Návrh a popis implementácie porovnávanej MapReduce aplikácie....46
5.3.2 Popis experimentu....46
6 Záver....48
7 Vízie a ďalšie rozšírenia....49
8 Literatúra....49
Príloha 1....51
Príloha 2....52
Zoznam príloh....53

Zoznam tabuliek ilustrácií

Zoznam ilustrácií

Obrázok 2.1: jednoduchá komunikácia....7
Obrázok 2.2: blokované a neblokované odosielanie správy....8
Obrázok 2.3:blokujúca komunikácia....8
Obrázok 2.4: Synchrónna neblokujúca komunikácia, zdroj:[8], obr. 2.15....9
Obrázok 2.5: Asynchrónna neblokujúca komunikácia, zdroj: [8], obr. 2.17....9
Obrázok 2.6: Nespoľahlivá komunikácia....10
Obrázok 2.7: Spoľahlivá komunikácia....10
Obrázok 2.8: Centralizované riadenie....12
Obrázok 2.9: Decentralizovaná architektúra....12
Obrázok 2.10: Vektorové hodiny ([11], Figure 3)....15
Obrázok 2.11: Centralizovane riadené vzájomné vylúčenie....16
Obrázok 3.1: Paralelné spracovanie [17]....22
Obrázok 3.2: Paralelné a zreťazené spracovanie [17]....23
Obrázok 3.3: Zreťazené spracovanie [17]....23
Obrázok 3.4: MapReduce....23
Obrázok 4.1: Inicializácia systému....26
Obrázok 4.2: Odľahčenie uzlu....27
Obrázok 4.3: XML konfigurácia - diagram tried....29
Obrázok 4.4: Inicializácia....30
Obrázok 4.5: Presun súboru s viacnásobným prístupom....33
Obrázok 4.6: Vyvažovanie záťaže medzi dvoma uzlami....35
Obrázok 4.7: Diagram tried pre prenos súborov....36

Zoznam tabuliek

Tabuľka 1: Spracovanie CC-2014-49, dátum:24.4.2015....44
Tabuľka 2: Spracovanie CC-2014-49, dátum: 9.5.2015....45
Tabuľka 3: Spracovanie dát na athena2, athena4, athena6....45
Tabuľka 4: Spracovanie 30GB dát na pc athena 2, 4, 6....46

1 Úvod

V dnešnom svete je často potrebné pracovať s veľkým množstvom dát, ktorých spracovanie trvá nezanedbateľný čas. Jednou z možností, ako sa vysporiadať s problémom výkonu, je rozdeliť prácu viacerým výpočtovým zdrojom. Žiaľ, aj jednoduché paralelné spracovanie na skupine počítačov má svoje úskalia:

- zdĺhavé spúšťanie spracovávania na jednotlivých počítačoch, ak ich je vyšší počet
- rozdielna doba spracovania jednotlivých počítačov zapríčinená preťažením jedného počítača, prípadne hardvérovými problémami
- zlyhanie počítača a jeho detekcia

Aktuálne technológie ponúkajú možnosti automatizácie celého procesu, pomocou ktorých je možné vytvoriť nový systém, ktorý zabezpečuje distribúciu spracovávania na mnohých počítačoch a vyžaduje iba minimálnu kontribúciu človeka. Systém, ktorý tvorí novú vrstvu medzi užívateľom a samotným spracovávaním. Systém je tvorený každým jedným počítačom, ktorý doň patrí. Takýto systém sa nazýva distribuovaný systém.

Hlavným cieľom práce je navrhnúť a implementovať distribuovaný systém, ktorý má spracovať dáta užívateľom stanoveným procesom v čo najkratšom možnom čase. Systém má za úlohu rozhodovať, ktoré dáta sa budú spracovávať na ktorých počítačoch.

Práca ukazuje základné problémy vyskytujúce sa v distribuovaných systémoch a možnosti ich riešenia. Medzi tieto problémy patrí komunikácia jednotlivých počítačov a zlyhania tejto komunikácie, dohodnutie počítačov na istom výsledku, meranie času, prípadne mechanizmus rozhodujúci poradie udalostí, udržanie požadovaného modelu konzistencie a detekcia zlyhaní. Sú popísané dva prístupy riadenia – centralizované a decentralizované.

Práca popisuje návrh a implementáciu distribuovaného systému spracovávajúceho rozsiahle textové dáta. Je popísaný centralizovaný prístup riadenia, keď sa všetky počítače podriaďujú rozhodnutiu hlavného. Je ukázaný spôsob vyvažovania záťaže buď prenesením súboru cez sieť, alebo jednoduchým odobratím z kolekcie úloh jedného počítača do kolekcie úloh počítača druhého, ak majú oba počítače prístup k danému súboru. Práca popisuje možnosti spracovania dát buď viacerými konkurentne bežiacimi procesmi, ktoré spracovávajú jeden súbor, alebo jedným procesom, ktorý komunikuje so systémom a následne spracováva súbory. Voľba spôsobu paralelizácie je potom prenechaná tomuto procesu. Ďalej je popísaná komunikácia medzi počítačmi na viacerých vrstvách. Je popísané riešenie zabezpečujúce detekciu chýb komunikácie a zachovanie modelu konzistencie. Je popísaný spôsob prenosu správ po sieti, ich serializácia a deserializácia. Následne sú popísané vykonané experimenty – spracovanie veľkého množstva dát a porovnanie so systémom Hadoop.

2 Distribuovaný systém

Internetová stránka [1] definuje distribuovaný systém ako program alebo sada programov, ktoré pre svoj beh využívajú viac ako jeden výpočtový zdroj. Metódy distribuovaných výpočtov pokrývajú široké spektrum, od viacvláknových aplikácií, cez aplikácie, ktoré pre svoj beh využívajú jeden systém, napríklad sieťový klient a server na jednom počítači, až po aplikácie, kde klientský program a server beží na počítačoch často veľmi vzdialených, príkladom sú webové aplikácie.

Kniha [2] definuje distribuovaný systém ako kolekciu nezávislých počítačov, ktoré sa javia užívateľovi ako jeden súdržný systém.

Časté skúmané vlastnosti sú: škálovateľnosť, výkon, priepustnosť, dostupnosť a odolnosť proti chybám. Jednotlivé vlastnosti sú definované v nasledujúcej kapitole.

Za cieľ distribuovaného systému možno považovať sprístupnenie viacero od seba nezávislých zdrojov, aby sa javili ako jeden.

2.1 Vlastnosti distribuovaného systému

2.1.1 Škálovateľnosť

Schopnosť systému zvládnuť narastajúce množstvo práce, prípadne schopnosť prispôsobenia sa potrebnému rastu. [3]

Škálovateľnosť možno rozlíšiť na niekoľko kategórií: [4]

- Funkčná škálovateľnosť: schopnosť vylepšiť systém pridaním nových funkcií bez zbytočne veľkej réžie.
- Administratívna škálovateľnosť: schopnosť zdieľania systému pre zvýšený počet užívateľov.
- Geografická škálovateľnosť: schopnosť udržovať výkon a efektivitu pri fyzickom rozšírení systému do väčšej oblasti.
- Záťažová škálovateľnosť: schopnosť efektívne využiť zdroje pri väčšej záťaži.
- Generačná škálovateľnosť: možnosť rozšírenia systému využívaním nových komponent.

Systém môže byť škálovateľný vo viacerých kategóriách. Je tiež nutné poznamenať, že zvýšenie miery škálovateľnosti jednej kategórie môže ovplyvniť mieru škálovateľnosti inej kategórie. Ako príklad je možné uviesť systém, ktorému je pridaný vysokorýchlostný bezdrôtový komunikačný modul pre nové možnosti komunikácie medzi jednotlivými počítačmi zlepšenie záťažovej

škálovatelnosti. Tento modul môže mať obmedzený dosah iba na niekoľko metrov, naviac v okolí nesmú byť fyzické prekážky. Tento spôsob ovplyvní geografickú škálovateľnosť.

2.1.2 Výkon

Výkon je definovaný ako množstvo práce za jednotku času. Výkon systému alebo aplikácie je možné merať rôznymi spôsobmi. Ak sa jedná o distribuovaný databázový systém, výkon môže byť definovaný ako počet spracovaných transakcií za jednotku času. Ak má systém za úlohu spracovávať súbory, výkon je možné definovať ako počet spracovaných bitov za sekundu, prípadne za aký dlhý čas sa spracuje jeden súbor. Táto vlastnosť je dôležitá pre každý systém, pretože každá operácia, ktorú systém vykonáva má skončiť v konečnom čase. Často krát je ale potrebné, aby operácia bola dokončená čo najskôr pri zachovaní ďalších požadovaných parametrov systému. Ako príklad môže poslúžiť databázový systém, ktorý po dokončení transakcie (operácie), má byť v konzistentnom stave.

2.1.3 Priepustnosť

Priepustnosť je definovaná ako počet úloh, ktoré môžu byť spracované za dlhšiu časovú periódu. [5]

2.1.4 Dostupnosť

Dostupnosť je definovaná ako čas, počas ktorého je systém vo fungujúcom stave.[6] Dostupnosť možno vypočítať nasledovne: *dostupnosť = funkčný čas / (funkčný čas + nefunkčný čas).*

2.1.5 Odolnosť proti chybám

Odolnosť proti chybám je vlastnosť systému pracovať správne v prípadne výskytu chyby. [6]

Pre zabezpečenie tejto vlastnosti je nutné definovať správne správanie systému a v závislosti od tohto predvídať rôzne možnosti zlyhania a výskytu chýb. Na tieto chyby je potrebné správne reagovať, aby sa systém správal podľa definovaných pravidiel.

2.2 Model distribuovaného systému

O distribuovanom systéme možno prehlásiť:

- program je vykonávaný konkurentne na nezávislých výpočtových uzloch [6]
- výpočtové uzly spolu komunikujú správami, ktoré môžu byť nedoručené
- výpočtové uzly nemajú zdielanú pamäť alebo zdielaný systém zámku [6]

Výpočtový uzol vykonáva algoritmus využívajúci lokálne zdroje. Uzol zabezpečuje: [6]

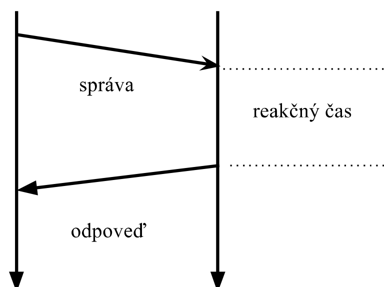
- vykonanie programu
- zápis dát do dočasnej alebo trvalej pamäte
- lokálny čas

Na každom výpočtovom uzle môže nastať chyba. Chyba môže byť trvalá, príkladom môže byť výpadok napájania a tým pádom zlyhanie vykonávaného programu, ale aj dočasná. Pod dočasnou chybou si je možné predstaviť dočasnú nedostupnosť internetového pripojenia.

Jadro distribuovaného systému tvoria výpočtové uzly, na ktorých sa vykonáva požadovaný algoritmus. Jednotlivé uzly sa môžu dostať do chybného stavu a tieto stavy je potrebné predvídať, a naplánovať príslušné reakcie po ich detekcii.

2.3 Komunikácia

Väčšina distribuovaných systémov potrebuje, aby jednotlivé výpočtové uzly (ďalej uzly) spolu komunikovali pre zabezpečenie rôznych vlastností. Komunikáciu môžeme chápať ako proces prenosu informácií. Obr. 2.1. Pre prenos informácií sa využíva správa. Správa je kolekcia objektov pozostávajúca z hlavičky o pevnej dĺžke a premenlivej dĺžky tela. Správa môže byť spracovaná procesom a doručená do cieľa. [7] Aby bola komunikácia efektívna, je potrebné definovať pravidlá pre správy, aby mohli byť riadne spracované prijímateľom.

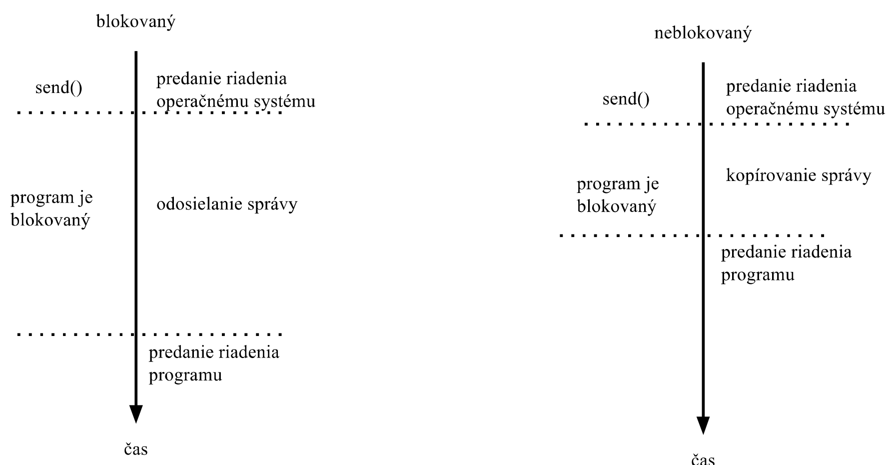


Obrázok 2.1: jednoduchá komunikácia

2.3.1 Blokovaná a neblokovaná komunikácia

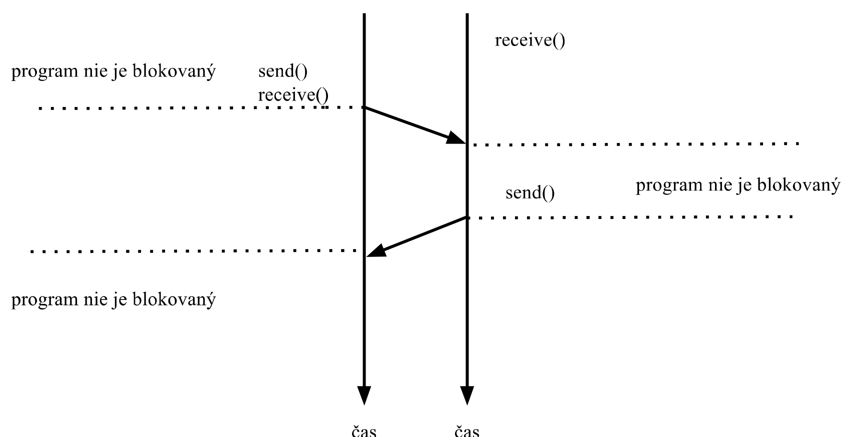
Dôležitá vlastnosť komunikácie je oneskorenie. Jednak trvá istý čas, kým je správa spracovaná nižšími vrstvami, odoslaná a doručená, ale aj samotná odpoveď musí prejsť nielen týmto procesom, ale prv musí prijímateľ zareagovať na doručenú správu - reakčný čas a nakoniec odpovedať. Komunikáciu rozlišujeme na blokujúcu a neblokujúcu. O neblokujúcej hovoríme vtedy, ak jej vykonanie nezdržuje vysielateľa, inak hovoríme o blokujúcej. [7] Tiež môžeme skúmať dve kritéria

tohto rozdelenia. Prvým je blokovanie operačným systémom, keď pri neblokujúcej variante je program blokovaný iba na čas potrebný k skopírovaniu správy do pamäte a blokujúcej variante, kedy je program blokovaný na čas, kým nieje odoslaná správa. (7 s. 38) Tento rozdiel ilustruje obrázok 2.2.



Obrázok 2.2: blokované a neblokované odosielanie správy

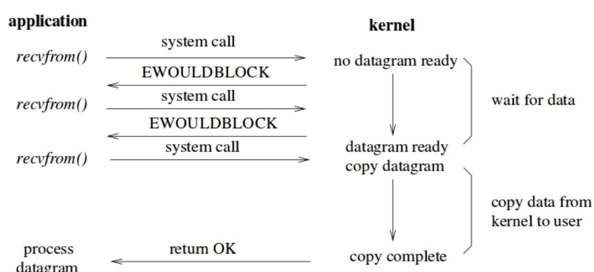
Druhým kritériom je samotná komunikácia medzi dvomi uzlami. O blokujúcej komunikácii hovoríme vtedy, keď jeden uzol po odoslaní čaká na prijatie odpovedi – obr. 2.3. Prípadne je to možné brať z opačného pohľadu, keď prijímateľ zavolá funkciu na prijatie správy receive() a je blokovaný, pokiaľ nieje nejaká správa prijatá. [7]



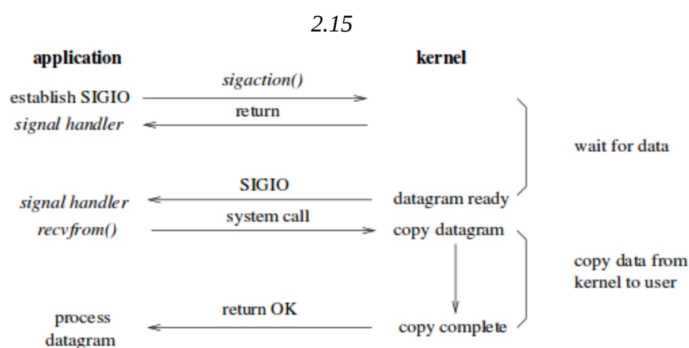
Obrázok 2.3:blokujúca komunikácia

O neblokujúcej komunikácii hovoríme vtedy, keď pri prijímaní správy je programu predaná prijatá správa alebo informácia, že žiadna správa nebola doposiaľ prijatá – v unix systéme chybová

správa EWOULDBLOCK [8]. Viď obr. 2.4. Program tiež môže nastaviť funkciu, prípadne obslužnú rutinu schránke, ktorá sa vykoná v prípade, keď schránka obdrží správu – v unix systéme obslužná funkcia reagujúca na signál prijatia správy do schránky. [8] Obr. 2.5 .



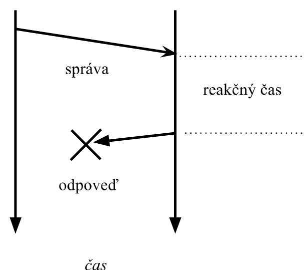
Obrázok 2.4: Synchrónna neblokujúca komunikácia, zdroj:[8], obr. 2.15



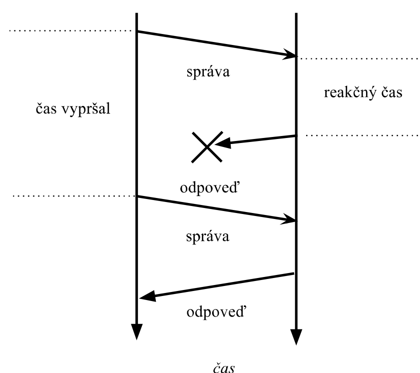
Obrázok 2.5: Asynchrónna neblokujúca komunikácia, zdroj: [8], obr. 2.17

2.3.2 Spoľahlivosť komunikácie

Pri komunikovaní je kritické, aby bola správa doručená. Pri prenose informácií však môže nastať chyba, napríklad správa sa počas prenosu nenávratne poškodí, prijímateľ sa dočasne stane nedostupným – výpadok internetového pripojenia. Prípadne na strane prijímateľa sa vyskytla vážna chyba. Ilustruje to obrázok 2.6. Pre zvýšenie odolnosti voči chybám je možné opakovane odosielať správu pri vypršaní maximálneho predpokladaného reakčného času, podobne ako je znázornené na obrázku 2.7. V spoľahlivej internej medzi-procesorovej komunikácii odosielateľ predpokladá možnosť zlyhania [7] a v závislosti od komunikačného protokolu znovu odošle poslednú odosielanú správu v danej komunikácii po vypršaní stanoveného času. Pri návrhu systému je potrebné uvažovať, ktorá vrstva by mala byť zodpovedná za spoľahlivosť. Je možné využiť protokol TCP a prenechať zodpovednosť operačnému systému, prípadne využiť protokol UDP a zodpovednosť ostane samotnej aplikácii.



Obrázok 2.6: Nespoľahlivá komunikácia



Obrázok 2.7: Spoľahlivá komunikácia

2.3.3 Architektúra siete

Sieť je možno rozdeliť na 7 vrstiev podľa referenčného modelu OSI. Stručná charakteristika jednotlivých vrstiev podľa [8]:

- Fyzická vrstva – definuje vlastnosti samotnej linky, napríklad úrovne napätia.
- Linková vrstva – popisuje prenos dát na jednej linke, adresovanie a vytváranie rámcov.
- Sieťová vrstva – definuje adresovanie a smerovanie dát.
- Transportná vrstva – garantuje spoľahlivý prenos medzi jednotlivými koncovými uzlami. Vrstva implementuje spojové a nespojové protokoly, ktoré sú podľa typu prenosu, spôsobu fragmentácie dát rozdelené do piatich kategórií.
- Relačná vrstva – slúži k udržiavaniu relácií medzi komunikujúcimi aplikáciami.
- Prezenčná vrstva – zaisťuje zobrazenie dát medzi rôznymi architektúrami a aplikáciami.
- Aplikačná vrstva – definuje užívateľské procesy a aplikácie komunikujúce po sieti.

Distribuovaný systém zväčša beží na operačnom systéme, ktorý zabezpečuje prvé 4 vrstvy a aplikácia má možnosť zvoliť transportný protokol. Na výber býva TCP alebo UDP protokol. Operačný systém môže naviac poskytnúť raw schránku a aplikácia môže implementovať vlastný protokol, kde je možné použiť vlastnú hlavičku v štvrtej vrstve. Pre bežné potreby ale stačia prvé dva spomenuté protokoly.

2.3.3.1 TCP oproti UDP

Základná charakteristika UDP:

- nespoľahlivá komunikácia [8]
- relatívne malá hlavička [13]
- minimálna réžia [8]

Základná charakteristika TCP:

- spoľahlivá komunikácia, ktorú zabezpečuje operačný systémoch [8]
- väčšia hlavička ako pri UDP [14]
- vyššia réžia [8]

Pri jednoduchých správach, často relatívne malých, ktoré sú informatívne a môžu mať pevnú dĺžku, je vhodné použiť protokol UDP. Ak je nejaká správa nedoručená z rôznych dôvodov, aplikácia je schopná po vypršaní času, ktorý je stanovený pre odpoveď (ak je vyžadovaná) znovu odoslať správu. Ak by bol využitý TCP, prinieslo by to stratu v rýchlosti komunikácii, kvôli zvýšenej réžii pre zabezpečenie spoľahlivosti a nadväzovaní spojenia. Pre prenos súborov, prípadne väčšieho objemu dát, je vhodnejšie použiť TCP protokol. Spoľahlivosť prenosu je zabezpečená operačným systémom a zvýšená réžia pri časovo dlhšom prenose je zanedbateľná.

2.4 Konsenzus

Viaceré procesy dosiahnu konsenzus, ak sa zhodnú v rozhodovaní. Musia platiť tieto pravidlá: [6]

- Každý správny proces rozhodne o rovnakej hodnote
- Každý správny proces rozhodne o najviac jednej hodnote. Ak rozhodne práve o jednej, rozhodne o nej stanoveným spôsobom.
- Každý správny proces rozhodne o hodnote v konečnom čase.
- Ak každý správny proces navrhne rovnakú hodnotu H, potom každý správny proces rozhodne o hodnote H.

Problém konsenzu je možno aplikovať na veľa oblastí distribuovaného systému. Voľba vodcu – koordinátora, vzájomné distribuované vylúčenie, udržanie replikácií v konzistentnom stave a iné. [6]

2.5 Architektúra distribuovaného systému

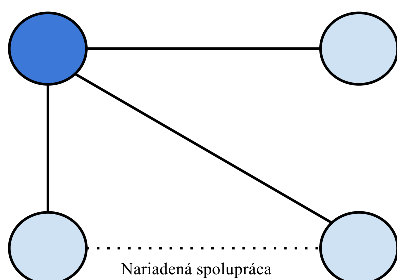
Architektúra systému popisuje konceptuálny model, ktorý definuje štruktúru, správanie a hlbší pohľad na systém. [9]

2.5.1 Centralizované riadenie

Systém riadený z centrálneho uzlu, obr. 2.8. V tomto prípade je logika a riadenie systému umiestnená na jedinom uzle – hlavný uzol, tzv. master, ktorý riadi uzle typu slave – otrok. Tento spôsob prináša mnohé výhody, napríklad jednoduchosť systému, z čoho vyplýva jednoduchší návrh a implementácia. Tiež je jednoduchšia správa systému, keďže všetky dôležité rozhodnutia sa dejú na jednom mieste. Otrocké uzly môžu spolupracovať až po nariadení hlavným uzlom, ktorý rozhodne, čo sa má

dosiahnuť a otrocké uzle dospejú k rozhodnutiu stanoveným procesom. Príkladom môže byť systém, ktorý má vykonať definované úlohy a odľahčenie príliš zaťaženého uzlu, keď hlavný uzol nariadi rýchlemu uzlu, aby odľahčil pomalý – prevezme časť jeho fronty úloh, ktoré má vykonať. Rýchlejší uzol teda prevezme úlohy z fronty dopredu stanoveným spôsobom.

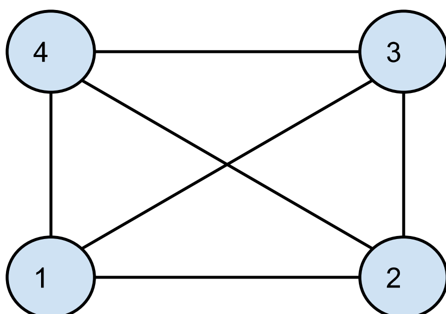
Medzi nevýhody patrí najmä možnosť zlyhania hlavného uzlu. S touto možnosťou je nutné počítať a definovať ďalšie správanie. Rolu môže prebrať iný uzol, prípadne ostatné uzly dokončia pridelenú prácu a ostanú neukončené, čakajúce na ďalší príkaz, často krát príkaz ukončenia. V niektorých prípadoch teda zlyhanie riadiaceho uzla nemusí byť problémové. Treba však zariadiť, aby sa systém správal správne. Príkladom môže byť opäť systém, ktorý spracováva zadané úlohy. Hlavný uzol neočakávane skončí uprostred presúvania úlohy z fronty jedného uzlu do fronty úloh uzlu druhého. Nesmie sa stať, aby bola úloha odobraná z fronty prvého uzlu a nepridaná do fronty druhého.



Obrázok 2.8: Centralizované riadenie

2.5.2 Decentralizované riadenie

O decentralizovanom rozhodovaní hovoríme vtedy, keď rozhodnutie je dosiahnuté spoločnou dohodou viacerých uzlov. Systém môže byť fyzicky rozdelený na niekoľko logicky rovnocenných častí, ale každá časť operuje nad vlastnou množinou dát. [2] Častá býva tzv. full-mash topológia, obr. 2.9.



Obrázok 2.9: Decentralizovaná architektúra

2.6 Čas

Čas je pre systém veľmi dôležitý. Podľa času je možné zoradiť operácie, ktoré sa majú vykonať, prípadne vykonali v minulosti. Meranie času je zväčša zabezpečované na počítačoch hardvérovým modulom, ktorý generuje jednotlivé impulzy pre zvyšovanie hodnoty vektoru, ktorý nám udáva logický čas od stanoveného počiatku. V module sa môže nachádzať kryštál, ktorý kmitá na istej frekvencii. Pomocou deličky je možné generovať impulzy v nami stanovenej frekvencii a merať čas. Problém ale nastáva, ak nie sú hodiny synchronizované a na každom počítači je iný nameraný čas. Toto môže byť pre distribuovaný systém kritické. Ak je systém postavený na centralizovanej architektúre, tak fakt, že hodiny sú oneskorené, nemusí robiť problém. V knihe [2] je predstavená situácia, keď pomocou programu make chceme skompilovať program. Ak máme zdrojový a objektový súbor, tak nás zaujíma iba fakt, či zdrojový súbor je vytvorený neskôr ako objektový. Zaujíma nás iba relatívny čas. Ak je systém postavený na decentralizovanej architektúre, tak v dôsledku nesynchronizovaných hodín môžu vzniknúť veľké problémy. Prvým príkladom môže byť operácia, ktorá nie je distributívna. Teda záleží na poradí vykonávania operácií. Ak na uzle A je vykonaná operácia nad dátami D a na uzle B rovnaká operácia nad D, je nutné rozhodnúť, ktorá sa má vykonať prvá. Ako druhý príklad, ktorý uvádza [2], si predstavme opäť program make a objektový súbor na uzle A, a zdrojový súbor na uzle B. Oba majú časovú značku podľa lokálnych hodín. Nie je možné porovnať lokálny čas uzlu A a lokálny čas uzlu B ak nie je istota, že hodiny sú synchronizované. Je teda potrebné, aby sa jednotlivé uzly dohodli na globálnom čase (pre systém), prípadne poradí jednotlivých udalostí. Je dôležité vedieť, že zdrojový súbor bol vytvorený neskôr ako objektový, prípadne, že operácia na uzle A bola zadaná skôr ako na uzle B. V mnohých prípadoch teda nie je potrebné poznať absolútny čas.

2.6.1 Lamportov logický čas

V článku [10] Leslie Lamport predstavil algoritmus pre udržiavanie logického času.
Čas je reprezentovaný množinou nezáporných celých čísel. Proces P_i vidí svoj lokálny a zároveň globálny čas (z jeho pohľadu) ako hodnotu C_i. Procesy dodržujú nasledujúce pravidlá: [2]

1 Pred odoslaním správy alebo vykonaním internej udalosti zvýši hodnotu C_i o nezápornú hodnotu. Často krát je táto hodnota 1 na každom procese pre jednoznačné identifikovanie udalosti na procese.

2 V čase odosielania správy m, proces pripojí časovú známku ts(m) rovnú hodnote C_i v čase odosielania k správe a príjemca C_j vykoná nasledujúce kroky:

2.1 ak je aktuálna hodnota C_j menšia ako ts(m), nastaví hodnotu C_j na ts(m)

2.2 vykoná krok 1

2.3 spracuje správu

2.6.2 Vektorové hodiny

Pomocou vektorových hodín možno určiť kauzalitu jednotlivých udalostí. Medzi dvomi udalosťami môžu byť nasledujúce vzťahy: [11]

- a → b znamená, že udalosť a predchádza udalosti b
- a ↛ b znamená, že udalosť a nepredchádza udalosti b
- a ↔ b znamená, že udalosti a,b sú konkurentné. Platí a ↛ b a zároveň b ↛ a

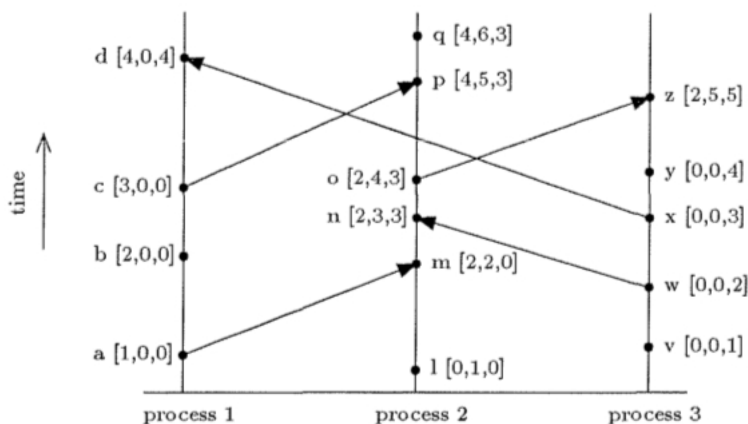
Majme množinu n procesov a každý môžeme identifikovať ako P_i. Vektorové hodiny môžu byť vytvorené, ak každý proces udržuje vektor VC_i s nasledujúcimi podmienkami: [2]

1. VC_i [i] je počet udalostí, ktoré sa vyskytli na procese P_i. Inými slovami VC_i [i] sú logické hodiny procesu P_i.
2. Ak VC_i [j] = k, potom P_i vie, že k udalostí sa vyskytlo na procese P_j. Je to teda znalosť i-teho procesu o lokálnom čase j-teho procesu.

Prvá podmienka je zabezpečovaná zvyšovaním VC_i[i] pri každom výskyte udalosti na P_i. Druhá podmienka je zabezpečovaná pripájaním vektoru k odosielaným správam. Nasledujúce pravidlá musia byť dodržané: [11]

1. Počiatočné hodnoty sú nulové.
2. Hodnota VC_i[i] je zvyšovaná aspoň raz pred každou atomickou udalosťou
3. K odosielanej správe správe je pripojená časová známka ts(m), ktorá je rovnaká ako VC_i.
4. Po prijatí správy m s časovou známkou ts(m) od procesu P_i, proces P_j vykoná dva kroky:
   1. Ak VC_j[i] <= ts(m)[i], potom $VC_j[i] \leftarrow ts(m)+1$
   2. $\forall k: VC_j[k] \leftarrow max(VC_j[k], ts(m)[k])$ a následne je možné spracovať správu.

Nech e_p je udalosť e vykonaná procesom p, potom Te_p je časová známka udalosti. Nech proces P je i-tým procesom, potom časová známka udalosti je Vc_i[i] v danom čase. (11) Obr. 2.10



Obrázok 2.10: Vektorové hodiny ([11], Figure 3)

Časove známky udalostí sú porovnávané nasledovne [11]: $e_p \rightarrow f_q \, ak : T_{e_p}[p] < T_{f_q}[p]$

2.7 Vzájomné vylúčenie

Jednotlivé procesy často potrebujú exkluzívny prístup do kritickej sekcie. V distribuovaných systémoch existuje viacero metód závislých na danej architektúre.

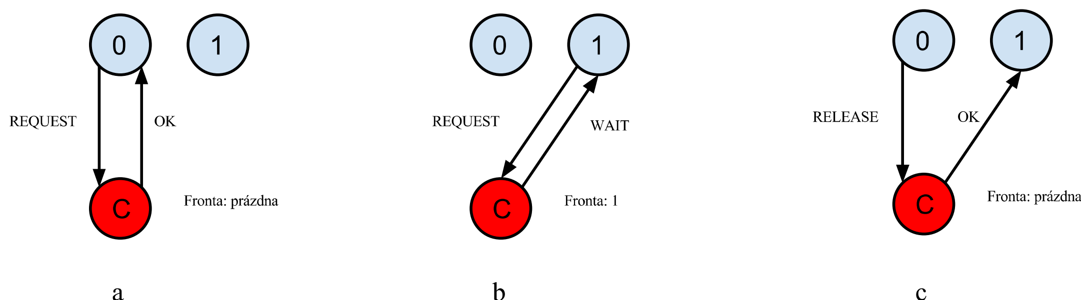
2.7.1 Centralizovaný algoritmus

Pri centralizovanej architektúre, keď rozhodovanie prináleží jednému uzlu, je jednoduché dosiahnuť výlučný prístup. Nech uzol, ktorý riadi prístup ku kritickým sekciám sa nazýva koordinátor. Ak uzol A chce požiadať o vstup do kritickej sekcie, pošle koordinátorovi správu REQUEST. Koordinátor v prípade, že uzol A môže do kritickej sekcie vstúpiť, odpovie správou OK. Obr. 2.11.a. [2] Ak uzol nemôže vstúpiť, požiadavok je vložený do fronty a sú tri možnosti ďalšieho postupu:

1. koordinátor uzlu neodpovie – uzol A čaká, pokým nedostane potvrdzovaciu správu OK [2]
2. koordinátor odošle správu WAIT – uzol A je informovaný o čakaní a vložený do fronty, obr. 2.11.b.
3. koordinátor odošle správu DENIED a uzol požiada opäť po uplynutí náhodného času.

Prípady sa od seba líšia možnosťou detekcie vážnej chyby, ktorá sa môže vyskytnúť u koordinátora. V prvom prípade je nezistiteľné, či koordinátor zlyhal a proces už nebeží. [2] Uzol stále čaká na správu OK, ktorá mu umožňuje prístup. V druhom a treťom prípade po vypršaní času je koordinátor považovaný za chybný.

Uzol, ktorý už nepotrebuje výlučný prístup do kritickej sekcie, pošle koordinátorovi správu RELEASE. V druhom prípade koordinátor vyberie z fronty ďalší požiadavok, ak nie je prázdna a existuje, a pošle žiadateľovi správu OK. [2] Obr. 2.11.c.



Obrázok 2.11: Centralizovane riadené vzájomné vylúčenie

V treťom prípade koordinátor čaká na najbližšiu správu REQUEST.

2.7.2 Decentralizovaný algoritmus

V prípade, že chceme zvýšiť odolnosť voči zlyhaniu jedného koordinátora, je možno vytvoriť viacerých koordinátorov, ktorý sa pri chybe obnovia v relatívne krátkej dobe. Následne je potrebný väčšinový súhlas pre vstup do kritickej sekcie. Ak je n koordinátorov, potom je potrebným n/2 potvrdzovacích správ. [2] Ak koordinátor nieje ochotný udeliť prístup, odošle správu DENIED. Pri vysokom počte procesov vyžadujúci prístup do rovnakej kritickej sekcie môže dôjsť k hladovaniu, keď je obtiažne získať väčšinový súhlas.

2.7.3 Ricartov Agrawalov distribuovaný algoritmus

V prípade, že je vyžadované, aby sa o výlučnom prístupe do kritickej sekcie dohodli jednotlivé uzly medzi sebou, je možné využiť distribuovaný algoritmus. Algoritmus funguje nasledovne. [19] Keď proces potrebuje vstúpiť do kritickej sekcie, vytvorí správu s identifikačným číslom procesu a lokálne poradie požiadavku v rámci systému. Následne odošle správu všetkým procesom, ktoré sa zúčastňujú rozhodovania. Odoslanie správy je považované za spoľahlivé. Proces udržiava najvyšše číslo požiadavku C. Keď proces prijme správu s požiadavkom pre vstup do kritickej sekcie s číslom požiadavku k a číslom procesu j, vykoná $C \leftarrow max(k, C)$ a následne má 3 možnosti:

1. Ak prijímateľ nevyužíva zdroje z kritickej sekcii a nepožaduje prístup do nej, odpovie správou REPLY.
2. Ak prijímateľ využíva zdroje z kritickej sekcii, neodpovie.

3. Ak požaduje prístup do kritickej sekcie, porovná C číslom k. Ak je číslo k menšie, odpovie správou REPLY, v opačnom prípade porovná číslo j s vlastným identifikačným číslom a podľa nastavenej politiky udeľovania priority pre celý systém odpovie správou REPLY alebo zaradí požiadavok do fronty a neodpovie.

Po odoslaní požiadavku proces čaká pokiaľ nedostane od každého odpoveď. Keď proces opustí kritickú sekciu, vyberie z neprázdnej fronty požiadavok. Následne zašle správu REPLY žiadateľovi.

2.8 Voľba koordinátora

V distribuovanom systéme pri spolupráci niekoľkých počítačov, je niekedy vhodné zvoliť koordinátora, ktorý má istú riadiacu úlohu. Nech jednotlivé uzly majú jednoznačné identifikačné číslo v rámci množiny uzlov. Toto číslo môže aj nemusí odrážať prioritu pri voľbe koordinátora. Algoritmus vytvorený Garbiarom-Molinom výberu je nasledujúci: [2]:

Keď proces P zistí, že koordinátor neodpovedá, vykoná nasledujúce kroky:

1. Proces P pošle správu ELECTION každému procesu s vyšším identifikačným číslom
2. Ak nikto neodpovie, proces P sa stane koordinátorom.
3. Ak niekto odpovie, pre proces P proces voľby končí.

V hociktorom momente môže proces obdržať správu ELECTION od iného uzlu s nižším identifikačným číslom. Ak sa tak stane, odpovie správou OK pre informovanie, že uzol pracuje správne. Následne pokračuje bodom 1 definovaným vyššie. Keď proces získa status koordinátora, informuje o tejto skutočnosti všetky ostatné uzle.

2.9 Distribuovaná zdielaná pamäť

Ak je systém tvorený z viacerých komunikujúcich počítačov, môžeme hovoriť, že na istej abstraktnej úrovni majú spoločnú zdielanú pamäť. Zdielanou pamäťou možno nazvať dáta, ku ktorým môže počítač v distribuovanom systéme pristúpiť pomocou lokálnych zdrojov alebo mu ich môže sprístupniť iný počítač pomocou siete. K zdielanej pamäti je možné pristúpiť pomocou operácie READ, ktorá načíta dáta a operácie WRITE, pomocou ktorej je možný zápis dát. Operácie je tiež možné interpretovať ako dvojicu invokácie a odpovede. Vykonanie operácie trvá konečný čas a vykonávanie operácií sa môže prelínať v čase.

2.9.1 Replikácie

V distribuovanom systéme často dochádza k replikovaniu dát a z toho vyplývajú dva problémy. [2] Prvým je manažment replikácií, ktorý zohľadňuje nie len umiestnenie uzlov s replikáciami, ale aj ako je obsah distribuovaný na uzle. Druhým je otázka, ako udržať replikácie konzistentné.

2.9.1.1 Dôvod replikovania dát

Hlavné dôvody pre vytváranie a udržovanie replikácií sú dva. [2] Prvým je spoľahlivosť. Ak jeden počítač zlyhá, jednoducho sa prepne na inú repliku, prípadne sa výpočet presunie na počítač, ktorý má prístup k replikovaným dátam.

Druhým dôvodom je výkon. Ak je systém geograficky distribuovaný v širokej oblasti a sprístupnenie zo vzdialenej oblasti by trvalo príliš dlhú dobu. Ďalším dôvodom môže byť zvyšujúci sa počet prístupov k dátam. Ak je v systéme príliš mnoho procesov, ktoré potrebujú pracovať s dátami spravované jedným serverom, môže to negatívne ovplyvniť výkon systému.

2.9.2 Model konzistencie

Model konzistencie definuje v akom poradí môže byť pristupované do zdielanej pamäti pomocou definovaných operácií. Najčastejšie sa jedná o operácie READ a WRITE. Zatiaľ čo tradičná definícia správnosti hovorí, že správne vykonanie operácií nad pamäťou je také, že každá operácia READ vráti hodnotu uloženú poslednou operáciou WRITE, definícia výrazu posledná operácia je pri konkurentnom prístupe nejasná. Je teda potrebné stanoviť presnú definíciu správnosti v takomto systéme. [2] V nasledujúcich kapitolách bude predstavený model silnej a slabej konzistencie.

2.9.2.1 Striktná konzistencia

Najstriktnejší model, zodpovedajúci jednoprocesorovému počítaču vyžadujúcemu, aby každá operácia READ danej lokácie vrátila hodnotu poslednej operácie WRITE danej lokácie. Je nutná implementácia zabezpečujúca globálny čas viditeľný každým procesom, čo môže byť náročné na zdroje.

2.9.2.2 Kauzálna konzistencia

Model slabej konzistencie vyžaduje, aby WRITE operácie, ktoré sú v kauzálnej relácii boli videné v rovnakom poradí všetkými procesmi, zatiaľčo konkurentné WRITE operácie, ktoré medzi sebou nemajú žiadny kauzálny vzťah je možné vidieť v inom poradí rôznymi procesmi. Kauzálna relácia je definovaná nasledovne: [2]

- Na procese sériové poradie udalostí definuje lokálne kauzálne poradie.

- WRITE operácia op1 kauzálne predchádza READ operáciu op2 na inom procese, ak op2 vracia hodnotu zapísanú op2.
- Tranzitívny uzáver predchádzajúcich dvoch relácií definuje globálne kauzálne poradie.

Je nutné dodať, že sa jedná o operácie rovnakej premennej, prípadne rovnakej lokácie v zdielanej pamäti.

2.10 CAP teorém

CAP teorém, vytvorený Ericom Brewerom, definuje 3 vlastnosti: [15]

- Konzistencia (consistency): všetky uzly vidia rovnaké dáta
- Dostupnosť (avalability): zlyhania jednotlivých uzlov nebránia pokračovaniu ostatných správne bežiacich uzlov
- Tolerancia rozdelenia (partition tolerant): systém pokračuje v behu napriek rozdeleniu systému z dôvodu nefunkčnej komunikácie medzi uzlami.

CAP teorém hovorí, že najviac dve vlastnosti môžu byť zabezpečené, tri nie. [15] Tento model je však zastaralý, pretože bolo vytvorených viacero modelov konzistencie. Preto aj sám Eric Brewer uviedol, že pravidlo 2 z 3 je zavádzajúce. Systémy sú často geograficky oddelené a so zlyhaniami samotnej siete sa musí počítať. Rovnako ale zlyhania nie sú časté a nieje potrebné vylúčiť konzistenciu alebo dostupnosť keď systém nieje rozdelený. V prípade, že je systém rozdelený, jednotlivé podsystémy môžu voliť medzi konzistenciou a dostupnosťou na základe operácií, ich vstupov a lokácií, nad ktorými operácie operujú. Nakoniec existuje viacero modelov konzistencie ako atomický a dostupnosť sa dá merať v percentách od nuly do sto. Systém by teda mal v prípade nerozdelenia systému poskytnúť perfektnú konzistenciu a čo najvyššiu dostupnosť a v prípade rozdelenia odhaliť rozdelenie, vstúpiť do bezpečného režimu, ktorý môže obmedziť niektoré operácie a po obnovení komunikácie začať proces, ktorý zabezpečí konzistenciu a napraví škody vzniknuté behom rozdelenia systému. [16]

2.11 Detektor zlyhania

Pri výpočte sa chyba môže vyskytnúť. Chyba môže byť fatálna, keď sa jedná o zlyhanie výpočtového zdroja – počítača, prípadne dočasná, keď je možná náprava, či už opätovným prevedením operácie alebo inou nápravou. Pravdepodobnosť výskytu chyby v systéme je priamoúmerná počte uzlov v systéme. V distribuovanom systéme ako v každom inom je potrené chyby a zlyhania objaviť a vykonať príslušné akcie. Riešenie nemusí byť triviálne, keďže zlyhať môže viacero vecí. Ako príklad

je možne uviesť, keď vzdialený počítač pracuje správne, ale chyba sa vyskytla v komunikačnej sieti, a teda nieje možné rozlíšiť, či zlyhali komponenty v sieti a je znemožnená komunikácia alebo zlyhal samotný počítač. [12]

Hlavné vlastnosti detektorov zlyhania sú úplnosť a presnosť.

2.11.1 Vlastnosti detektorov zlyhaní

Skúmané vlastnosti detektorov zlyhaní sú dve: úplnosť a presnosť. Úplnosť je ďalej možné rozdeliť na dva typy a presnosť na štyri. Detektor D je možné zredukovať na iný detektor D', ak existuje distribuovaný algoritmus, ktorý transformuje D na D'. Inými slovami každý problém, ktorý je riešiteľný detektorom D' je riešiteľný detektorom D. [12]

2.11.1.1 Úplnosť

Existuje čas, po ktorom je každý proces, ktorý zlyhal je permanentne podozrievaný správnym procesom.

Existujú dva typy úplnosti: [12]

- Silná úplnosť: každý proces, ktorý zlyhal je nakoniec permanentne podozrievaný zo zlyhania každým správnym procesom.
- Slabá úplnosť: každý proces, ktorý zlyhal je nakoniec permanentne podozrievaný zo zlyhania niektorým správnym procesom.

Silnú úplnosť nieje ťažké dosiahnuť. Príklad: nech každý proces permanentne podozrieva všetky ostatné procesy. Takýto detektor je zbytočný, je preto nutné zaistiť presnosť.

2.11.1.2 Presnosť

Existuje čas, po ktorom každý správny proces nieje podozrievaný zo zlyhania žiadnym správnym procesom.

Existujú dva typy presnosti: [12]

- Silná presnosť: správny proces nieje nikdy podozrievaný zo zlyhania žiadnym správnym procesom.
- Slabá presnosť: nejaký správny proces nieje nikdy podozrievaný zo zlyhania žiadnym správnym procesom.

Presnosť je považovaná za trvalú. Presnosť môže byť obtiažne implementovať, keďže detektor môže podozrievať správny proces a neskôr opraviť chybný predpoklad. Toto trvalá slabá presnosť nedovoľuje.

2.11.1.3 Eventuálna presnosť

Nemusí byť potrebné vyžadovať presnosť stále, ale aby v konečnom dôsledku bola uspokojená.

Rozlišujeme dva typy: [12]

- Silná eventuálna presnosť: existuje čas, po ktorom správne procesy nie sú podozrievané zo zlyhania žiadnym správnym procesom.
- Slabá eventuálna presnosť: existuje čas, po ktorom nejaký správny proces nieje podozrievaný zo zlyhania žiadnym správnym procesom.

2.11.2 Typy detektorov zlyhaní

Podľa vlastností možno klasifikovať detektory do týchto kategórií: [12]

- Perfektný detektor (P): detektor spĺňa podmienky silnej úplnosti a silnej presnosti.
- Eventuálny perfektný detektor (P): detektor spĺňa podmienky silnej úplnosti a eventuálnej silnej presnosti
- Silný detektor (S): detektor spĺňa podmienky silnej úplnosti a slabej presnosti.
- Eventuálny silný detektor (S): detektor spĺňa podmienky silnej úplnosti a eventuálnej slabej presnosti
- Slabý detektor (W): detektor spĺňa podmienky slabej úplnosti a slabej presnosti.
- Eventuálny slabý detektor (W): detektor spĺňa podmienky slabej úplnosti a eventuálnej slabej presnosti
- Detektor, ktorý spĺňa podmienky slabej úplnosti a silnej presnosti označujeme δ
- Detektor, ktorý spĺňa podmienky slabej úplnosti a eventuálnej silnej presnosti označujeme δ

Nech ≡ značí ekvivalenciu medzi jednotlivými kategóriami detektorov, potom P ≡ δ, S ≡ W, P ≡ δ, S ≡ W. Dôkaz je uvedený v [12] s. 572-576. Dôsledok tvrdenia spočíva v tom, že ak sa vyrieši problém silnej úplnosti štyroch typov detektorov, je vyriešený problém zvyšných štyroch detektorov.

3 Distribúcia výpočtu

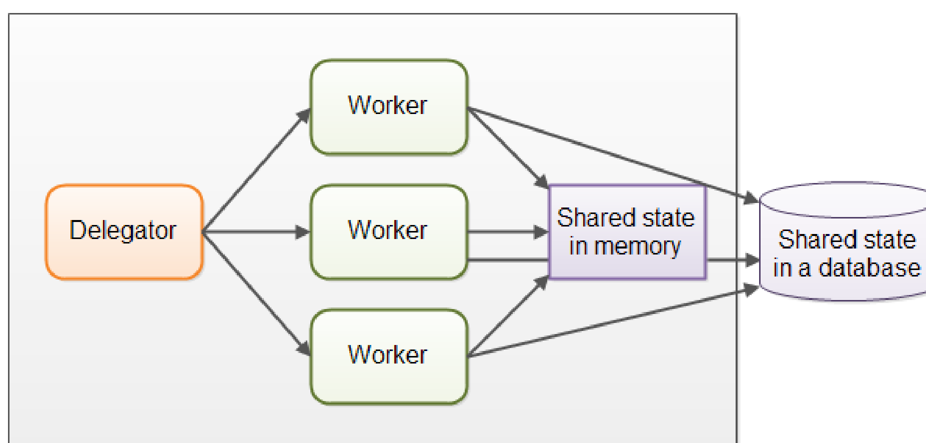
Pre zvýšenie výkonu systému je možné pridať viac výpočtových zdrojov a výpočet distribuovať. Viac úloh je teda vykonávaných konkurentne. Ak spolupracuje viacero počítačov, z ktorých ma každý k dispozícii lokálne zdroje, jedná sa o distribuovaný systém. Na jednom počítači môže byť výpočet distribuovaný medzi jednotlivé vlákna, prípadne procesory.

3.1 Konkurentné modely

Konkurentný model špecifikuje ako jednotlivé vlákna spolupracujú aby dokončili zadanú úlohu. [17] Na istej abstraktnej úrovni však možno prirovnať vlákno k procesu distribuovaného systému. Vlákno pracuje so zdielanou pamäťou a spolupracuje s ostatnými. Rovnako ako v distribuovanom systéme jednotlivé procesy. Je teda možné vidieť mnohé podobnosti. Napríklad vyvažovanie záťaže. [17]

3.1.1 Paralelní robotníci

V tomto modeli na začiatku delegátor inicializuje robotníkov, ktorý bežia v samostatných vláknach a spracovávajú jednotlivé úlohy. [17] Robotníci môžu pracovať s databázou – zdielanou pamäťou. Obr 3.1.



Obrázok 3.1: Paralelné spracovanie [17]

Existujú tri prístupy:

- delegátor inicializuje na začiatku robotníkov a každému pridelí jednu úlohu
- delegátor inicializuje na začiatku robotníkov a rozdelí medzi ne úlohy tak, aby bola záťaž vyvážená
- delegátor inicializuje na začiatku robotníkov a tí jednotlivé úlohy získavajú zo zdielanej pamäte – vyrovnanie záťaže je implicitné

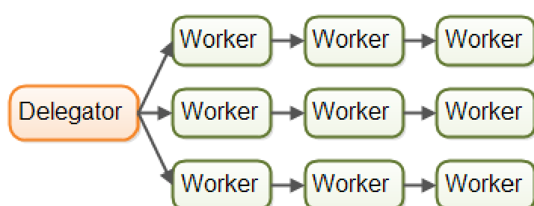
3.1.2 Zreťazení robotníci

V modeli je úloha rozdelená na viacero častí a každú časť vykoná jeden robotník analogicky s výrobou v továrni, keď robotník odvedie svoju časť práce na výrobok a posunie ho ďalšiemu robotníkovi, aby mohol na ňom spraviť svoju prácu. [17] Obr. 3.3.

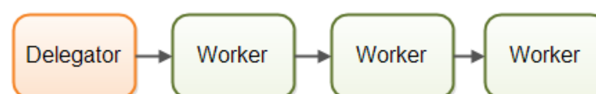
Jednotlivý robotníci si predávajú výsledok svojej práce vstupno-výstupnými operáciami, najčastejšie neblokujúcimi pre zvýšenie výkonu. Pri neblokujúcej sa nečaká na dokončenie operácie – počas toho sa môže robotník venovať ďalšej úlohe. [17]

3.1.3 Paralelne zreťazení robotníci

Model je kombináciou predchádzajúcich dvoch modelov. Existuje viac zreťazených robotníkov, ktorí plnia úlohy paralelne. Obr. 3.2. Úlohy sú rozdeľované ako paralelnom modeli a jednotlivé časti úloh spracovávané ako v zreťazenom modeli.



Obrázok 3.2: Paralelné a zreťazené spracovanie [17]

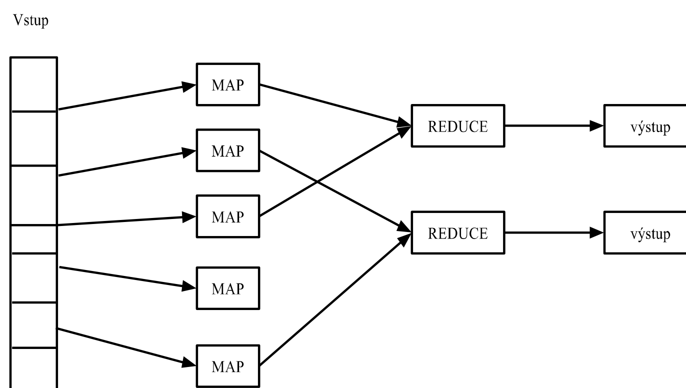


Obrázok 3.3: Zreťazené spracovanie [17]

3.2 MapReduce

MapReduce je programovací model pre spracovanie dát. Model podporovaný Hadoop v niekoľkých jazykoch ako Java, Python a C++. MapReduce programy sú často využívané pre spracovanie veľkých dát. [18]

MapReduce funguje rozdelením spracovania do dvoch fáz: map fáza a reduce fáza. Každá fáza má ako vstup a výstup páry kľúčov a hodnôt. Typy sú zvolené programátorom. [18] Viacero procesov vykonávajúce jednotlivé fázy môže byť vykonávaných konkurentne.



Obrázok 3.4: MapReduce

Ako príklad je možné uviesť spočítanie rovnakých riadkov v súbore. Každý riadok bude spracovaný jednou MAP funkciou, ktorej výstup bude dvojica riadok ako kľúč a číslo 1 ako hodnota. REDUCE funkcia získa ako vstup dvojicu kľúč a zoznam hodnôt, ktoré MAP funkcie tomuto kľúču pridelili (samé jednotky). REDUCE funkcia vytvorí súčet týchto hodnôt, teda spočíta rovnaké riadky a ako výstup bude dvojica riadok a súčet hodnôt. Následne sa zapíše do trvalej pamäte počet výskytov daného riadku v súbore. MapReduce model pozostáva z funkcií MAP a REDUCE.

4 Návrh a popis implementácie

4.1 Ciele systému

Aplikácia má za úlohu spracovať veľké súbory distribuované na viacerých výpočtových zdrojoch a paralelne na jednom výpočtovom zdroji. Úloha je všeobecná, je to teda proces definovaný užívateľom. Spúšťaný proces bude spracovávať jeden alebo viac vstupných súborov z užívateľom stanovenej množiny vstupných súborov. Procesy budú spúšťané paralelne, pričom každý spracuje jeden súbor alebo bude spustený jeden proces, ktorý bude komunikovať s aplikáciou a bude získavať cesty k súborom na spracovanie, a spôsob paralelizácie spracovania bude v réžii spúšťaného procesu.

Aplikácia má za cieľ vykonať úlohu a spracovať súbory čo najrýchlejšie. Príliš pomalé výpočtové uzle oproti iným musia byť identifikované a odľahčené odobratím adekvátneho počtu súborov pre spracovanie, a následné presunutie ich spracovania na iný výpočtový uzol. Má byť poskytovaná možnosť vrátenia výsledného súboru na pôvodný uzol po spracovaní na inom výpočtovom uzle.

Aplikácia má umožňovať vytvorenie základného konfiguračného súboru obsahujúci definície súborov na spracovanie. Ktoré súbory na ktorých počítačoch je treba spracovať určí užívateľ tak, že definuje zložky na jednotlivých počítačoch, v ktorých sa nachádzajú súbory na spracovanie. Aplikácia má tiež umožňovať vytvorenie základného konfiguračného súboru s výslednými súbormi po spracovaní.

Ďalším cieľom je dbanie na rozšíriteľnosť o ďalšiu potrebnú funkcionalitu.

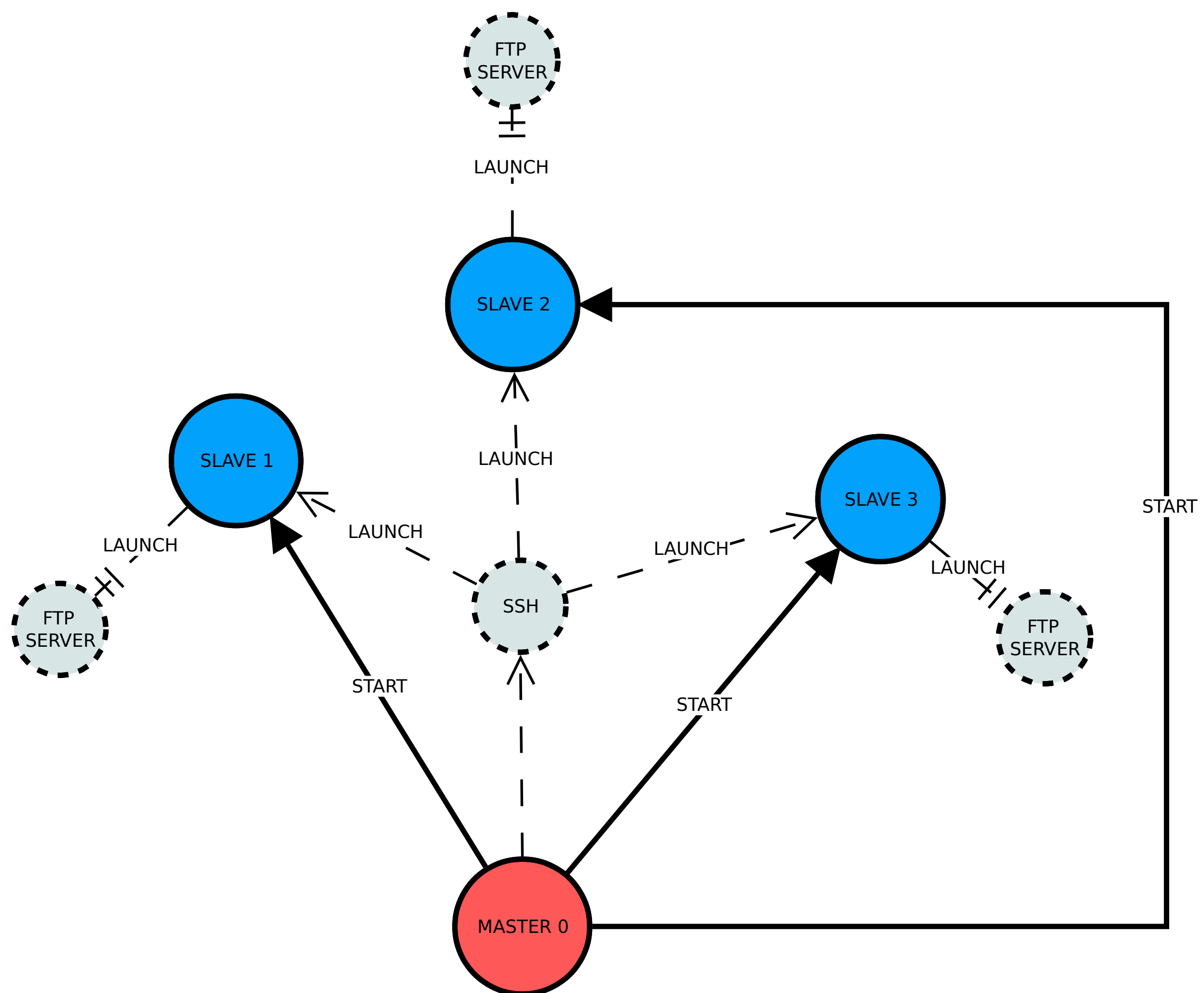
4.2 Model

4.2.1 Koncept

Architektúra riadenia systému bude centralizovaná, to znamená, že rozhodnutia a iniciovanie akcií bude vykonávať jeden uzol - master. Na začiatku master získa a spracuje konfiguráciu systému, ktorá definuje jednotlivé uzly a ich príslušné parametre, a množinu súborov s uzlami, ktoré majú k daným súborom lokálny prístup. Viac o konfigurácii v kapitole konfigurácia.

K súboru môže mať prístup viac uzlov. Príkladom môže byť replikácia súboru, kedy existujú dva súbory. Druhým príkladom je zdielané úložisko, keď sa jedná o jeden fyzický súbor. Z pohľadu aplikácie sú tieto dva prípady nerozlíšiteľné a aplikácia poskytuje metódu pre kontrolu ekvivalencie. Či bude metóda využitá, volí užívateľa. Kontrola je vykonaná porovnaním hash sumy.

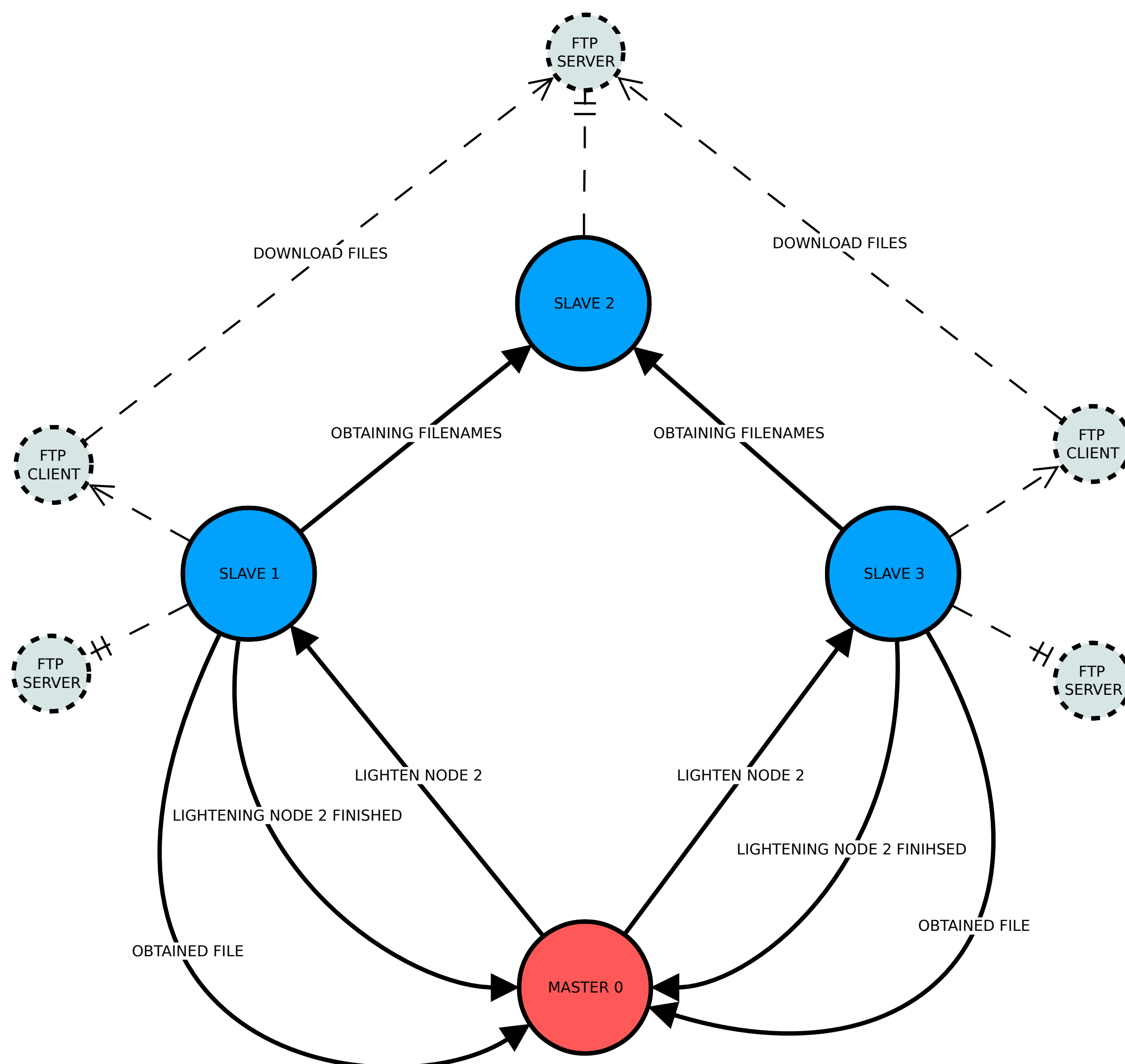
Master rozhodne, ktoré uzle budú spracovávať ktoré súbory a spustí na vzdialenom počítači rovnaký binárny súbor, ale s konfiguráciou pre otrocký uzol – slave. Všetky uzly v systéme spustia FTP server. Všetky podriadené uzly informujú uzol master o správnom spustení, ak sa tak nestane, master informuje užívateľa o zlyhaní a ukončí systém. V prípadne, že všetko prebehne správne, odštartuje vykonávane úloh. Master sa tiež môže zúčastňovať výpočtu podľa konfigurácie. Obr. 4.1.



Obrázok 4.1: Inicializácia systému

Po odštartovaní podriadené uzle informujú v pravidelnom intervale master uzol o odhadovanom čase ukončenia spracovávania. Na základe týchto informácií sa master snaží o dosiahnutie čo najväčšej priepustnosti odľahčovaním nadmerne zaťažených, prípadne najmenej výkonných uzlov. V prípade, ak pomalý uzol má vo fronte súbory, ku ktorým má lokálny prístup viac uzlov, master odoberie súbor z fronty pomalého uzlu a pridá ho do fronty uzlu rýchleho. Ak takéto súbory neobsahuje, tak prikáže výkonnejším uzlom aby pomalý uzol odľahčili.

Odľahčenie menej najvýkonnejšieho uzla spočíva získaní názvu súboru a jeho stiahnutia pomocou stanoveného protokolu. Po stiahnutí súboru a jeho umiestnení do fronty informuje master-a o získaní prístupu a pridaní súboru do vlastnej fronty úloh. Správou tiež informuje o odobratí súboru z fronty úloh pomalého uzla. Rýchlejší uzol pokračuje v získavaní súborov, pokým sa nedostane na podobný odhadovaný čas ukončenia. Obr. 4.2.



Obrázok 4.2: Odľahčenie uzlu

4.2.1.1 Dohoda uzlov

V systéme môže prebiehať viacero komunikácií súčasne. Uzol teda musí uchovávať stav každej svojej prebiehajúcej komunikácie, ak je to potrebné. Aby sa dospelo k určitému výsledku alebo k dohode, je často nutná viackroková komunikácia. Dohodu alebo operáciu, pre ktorú je potrebná komunikácia dvoch uzlov, je možné zabstraktniť na transakciu v systéme. Abstraktná trieda Transaction vyžaduje implementáciu nasledujúcich metód:

- execute() - metóda, ktorá je určená, aby bola z okolia volaná iba raz po vytvorení objektu a metóda vykoná operáciu
- putResponse(Object resp) – metóda pre spracovanie odpovede

- update() - metóda by mala byť volaná v pravidelných intervaloch zabezpečujúca znovuodoslanie správ ak vypršal pridelený čas na odpoveď
- canFinish() - ak metóda vráti true, transakcia je považovaná za ukončenú
- finish() - metóda volaná na úplný záver, ak metóda canFinish() vráti true
- isTimeout() - metóda vráti true, ak vypršal čas a transakciu je nutné ukončiť násilne
- cancel() - metóda volaná z okolia, ak isTimeout() vráti hodnotu true

Model je navrhnutý tak, aby spolu mohli komunikovať dva objekty typu Transaction. Správy musia obsahovať identifikátory, pomocou ktorých je možné jednoznačne identifikovať, pre ktorú transakciu je správa určená, prípadne vytvoriť novú transakciu (iniciátor je iný uzol). Služba TransactionService poskytuje správu pre udržanie viacero transakcií. Poskytuje metódy pre vytvorenie novej transakcie a následné vyvolanie metódy execute(), poskytuje metódu pre vloženie správy správnej transakcii – pomocou identifikátorov. Poskytuje metódu pre rozhodnutie, či daná prijatá správa patrí nejakej transakcii alebo je potrebné vytvoriť novú transakciu. Príkladom transakcie je správa, ktorej prijatie je treba potvrdiť. Odosielateľ vytvorí transakciu, ktorá odošle správu a pri prijatí odpovede ju je možné ukončiť. Uzol, ktorý správu prijal nemusí v tomto prípade inštanciu transakcie vytvoriť.

4.2.1.2 Model konzistencie

Nech P_i je proces bežiaci na výpočtovom uzle a M je množina súborov definovaná užívateľom určené pre spracovanie, MP_i je množina súborov určená pre spracovanie na P_i a MP_i' je množina spracovaných alebo práve spracovávaných súborov na P_i, potom je systém konzistentný práve vtedy, ak platí, že prienik všetkým množín MP_i a MP_i pre i od 1 do n je rovný množine M. N je počet procesov.

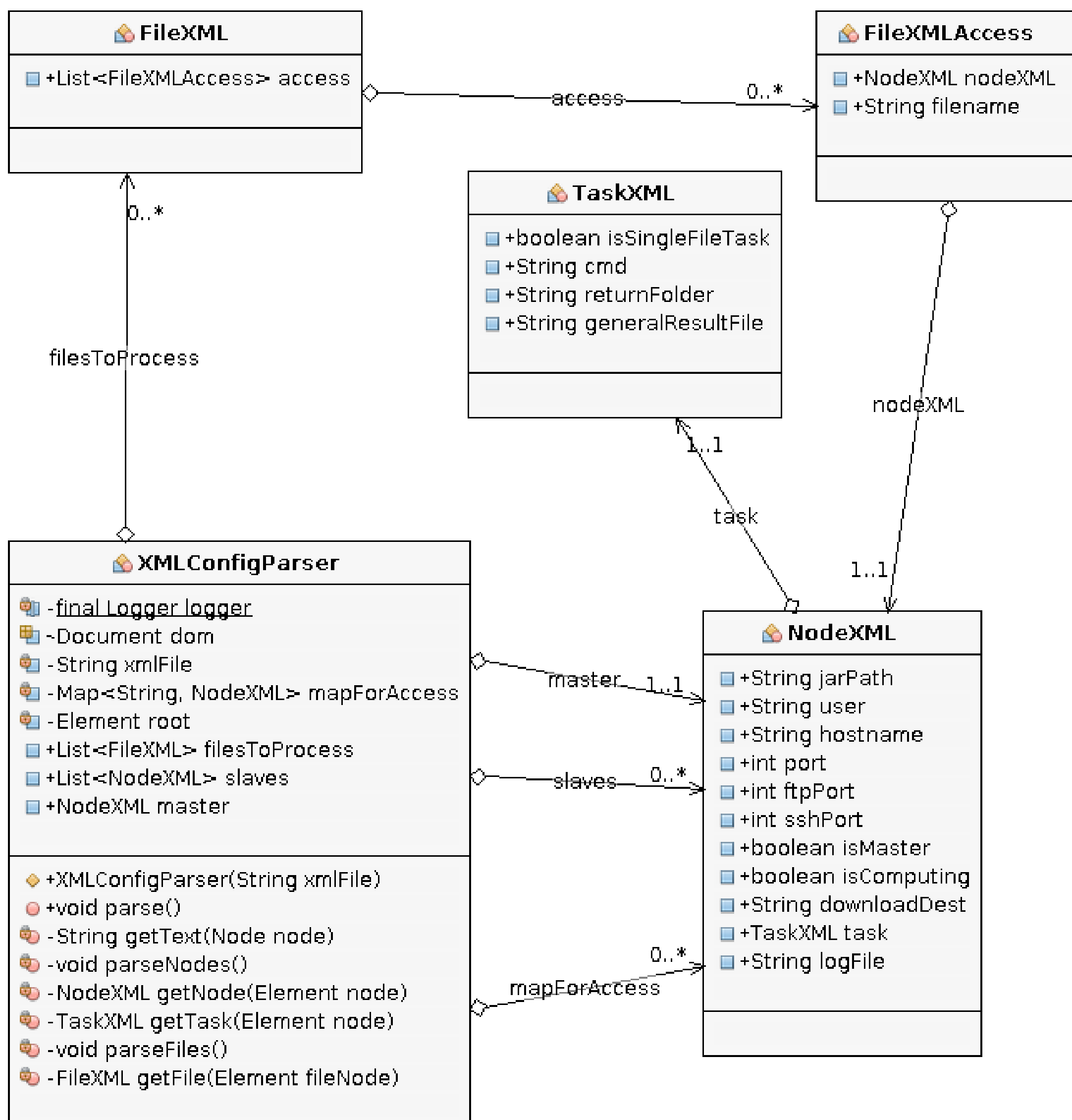
4.2.2 Popis implementácie

4.2.2.1 Inicializácia

Inicializačná fáza má niekoľko krokov.

Na začiatku je získaná konfigurácia systému z argumentov programu. Rozbor argumentov zabezpečuje trieda ArgParse a metóda parse, ktorá implementuje konečný automat na nastavenie jednotlivých verejných premenných inštancie. Ďalším krokom je získanie konfigurácie o jednotlivých uzloch a množine súborov, a prístupom k nim. Názov konfiguračného súboru je získaný ako argument aplikácie. Forma konfiguračného súboru je popísaná v kapitole konfigurácia. Formát súboru je xml

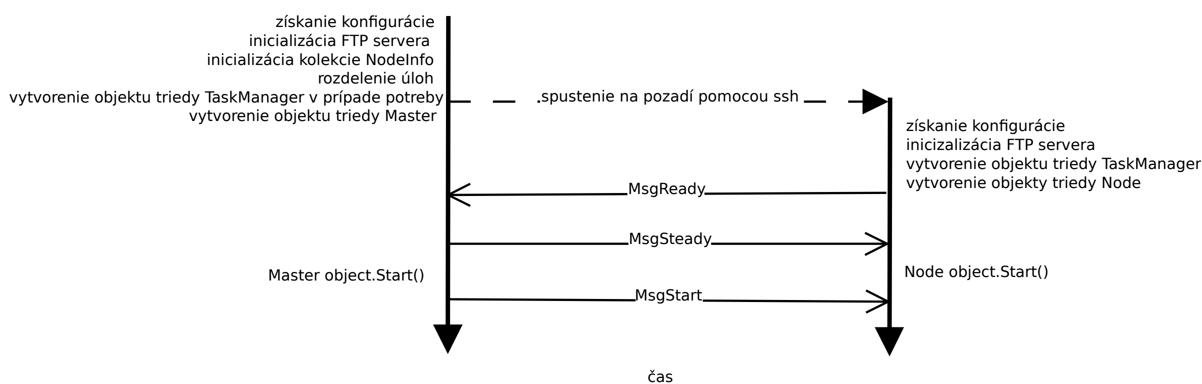
a extrahované informácie extrahuje inštancia triedy XMLConfigParser. Obr. 4.3. Obrázok je iba ilustratívny, kvôli veľkým rozmerom boli vynechané konštruktory.



Obrázok 4.3: XML konfigurácia - diagram tried

Druhá fáza je vytvorenie objektu typu MasterInitializer, ktorého metóda prepare inicializuje objekty reprezentujúce výpočtové uzly - NodeInfo. Každý slave uzol má unikátne identifikačné číslo, pričom master má 0. Vytvorí objekty FileObject reprezentujúce súbory, s informáciami o uzloch, ktoré k nim majú lokálny prístup. Pridelí ich jednotlivým výpočtovým uzlom. Snahou je, aby mal každý uzol rovnaký počet súborov na spracovanie, pričom súbor musí byť pridelený takému uzlu, ktorý má k nemu lokálny prístup. Ďalším krokom je inštanciovanie triedy Master, ktorá rozširuje

triedu Node a príslušnej podtriedy abstraktnej triedy TaskManager. Nasleduje spustenie programov na vzdialených počítačoch. Pomocou protokolu ssh2 MasterInitializer spustí na pozadí binárny súbor. Program bude nakonfigurovaný ako slave s parametrami zadanými užívateľom, adresou a číslom portu na ktorom načúva master a s názvami súborov, ktoré sú uzlu pridelené. Každý slave odošle uzlu master správu MsgReady, na ktorú MasterInitializer odpovie správou MsgSteady. Po prijatí správ o pripravenosti od všetkých uzlov, metóda prepare skončí a riadenie je prenechané objektu typu Master metódou start. Master každému odošle správu MsgStart pre začatie výpočtu. Ak nejaký uzol zlyhá v inicializačnej fáze, master ukončí beh všetkých uzlov a informuje užívateľa o zlyhaní. Ak bol v konfigurácii stanovený súbor pre zázanamy, užívateľ je schopný nájsť príčinu zlyhania. Inicializácia jedného slave uzlu je znázornená na obr. 4.4.



Obrázok 4.4: Inicializácia

4.2.2.2 Riadenie

Riadenie celého systému je prenechané master uzlu a slave uzly sa riadeniu podriaďujú. Konanie jadra slave uzlu spočíva v odosielaní správy MsgHello, ktorá obsahuje odhadovaný čas ukončenia a v reagovaní na správy, a v prípade potreby vytvorenie príslušnej transakcie, a prenechanie jej riadenie operácie.

Master uzol má za úlohu podľa získaných odhadovaných časov ukončenia rozhodovať o prípadnom presúvaní súborov. Po spracovaní súborov odošle MsgQuit správu všetkým uzlom, čím im dá príkaz pre ukončenie.

4.2.2.3 Vyvažovanie záťaže

Master uzol neustále hľadá najvýkonnejší a najmenej výkonný uzol, medzi ktorými je možné vykonať vyváženie záťaže. Uzle musia spĺňať nasledujúce podmienky.

Uzol, ktorý ma získať súbory:

- nesmie v čase získavať súbory od iného uzlu alebo byť odľahčovaný
- v posledných 10 minútach sa nesmela vyskytnúť závažná chyba počas FTP prenosu
- nesmú byť uzamknuté súbory inou transakciou (uzamknutie je voliteľné)
- nesmie byť detekovaný ako chybný

Uzol, ktorý má byť odľahčený:

- nesmie v čase získavať súbory od iného uzlu
- v posledných 10 minútach sa nesmela vyskytnúť závažná chyba počas FTP prenosu
- nesmú byť uzamknuté súbory inou transakciou (uzamknutie je voliteľné)
- nesmie byť detekovaný ako chybný

Po nájdení dvoch kandidátov najprv otestuje menej výkonný uzol, či môže byť vykonaná transakcia pre presun súborov s viacnásobným prístupom – musí existovať súbor s viacnásobným prístupom ku ktorému má lokálny prístup iný uzol, ktorého odhadovaný čas ukončenia je menší ako jedna hodina a 30 minút. Ak transakciu nieje možné vytvoriť, otestuje možnosť vykonania vyváženia záťaže medzi dvoma kandidátmi – výkonnejší uzol má odhadovaný čas ukončenia menší ako hodinu a 30 minút.

4.2.2.4 Transakcie

Model transakcie je navrhnutý pre komunikáciu, pri ktorej je potrebná aspoň jedna odpoveď. Správa obsahuje dva identifikátory nodeID a transactionID, pričom nodeID je identifikátor uzlu, ktorý transakciu zahajuje a transactionID je lokálne poradové číslo transakcie. Unikátne číslo poskytuje triedna metóda getNewTransactionID abstraktnej triedy Transaction. Prijímateľov objekt transakcie bude využívať tieto dva identifikátory. Mechanizmus zabezpečuje možnosť identifikovania príslušnej transakcie po prijatí správy.

Metóda execute odošle správu a v prípade nezískania odpovedi do 10 sekúnd odošle transakcia správu znovu (pri volaní metódy update). Toto platí pri väčšine transakcií. Ak do 60 sekúnd neobdrží odpoveď a metóda isTimeout vráti hodnotu true, transakcia je stornovaná metódou cancel. Zmieňované princípy platia pre všetky transakcie okrem transakcie pre vyrovnanie záťaže medzi dvoma uzlami.

Ak sa jedná o jednoduchý príkaz, ktorý vyžaduje iba potvrdzovaciu správu MsgAck, tak je transakcia implementovaná príslušnou podtriedou triedy Transaction. Potvrdzovacia správa obsahuje rovnaké identifikátory nodeID a transactionID. Transakcie tohto typu sú: príkaz pre odľahčenie konkrétneho uzlu, informovanie o skončení odľahčovania, informovanie o získaní prístupu k danému

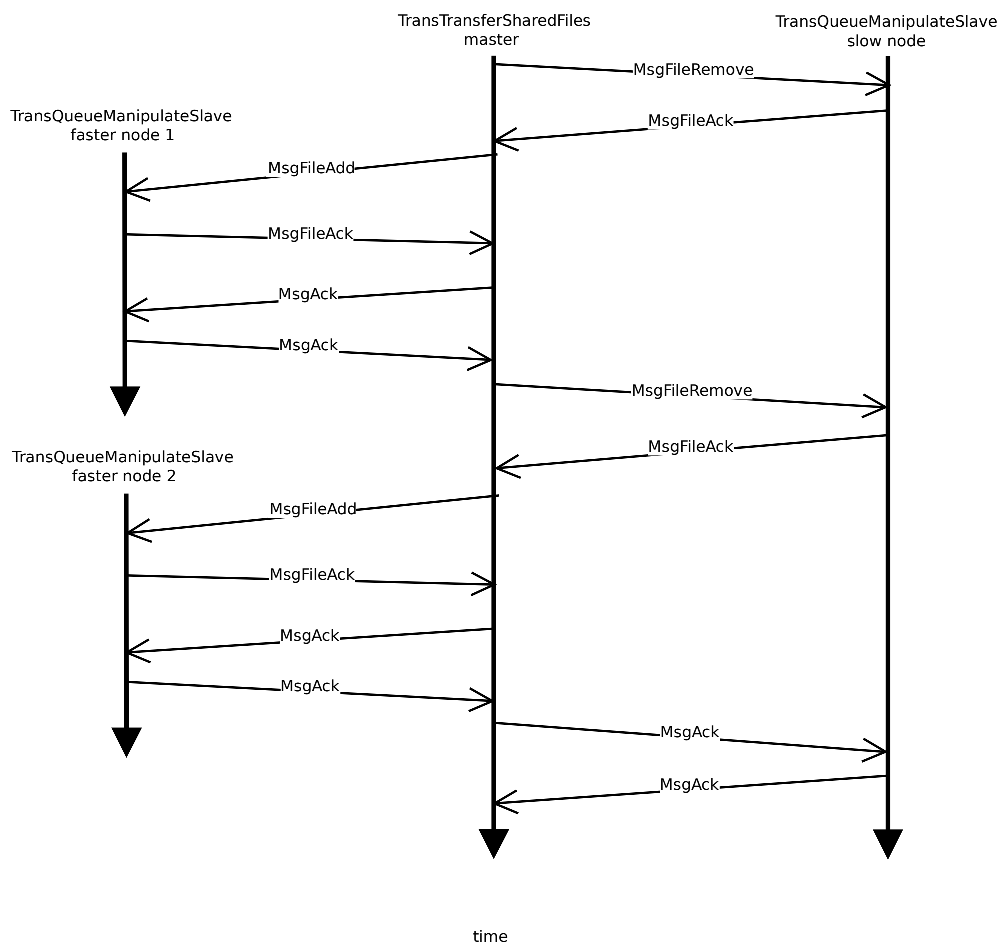
súboru, informovanie o výsledného súboru a jeho umiestnení a príkaz na zrušenie transakcie daného typu od konkrétneho uzlu.

Presun súboru s viacnásobným prístupom

Odľahčenie uzlu o súbory, ktoré majú viacnásobný prístup, implementuje trieda TransTransferSharedFiles. Využíva pre to správy MsgFileRemove a MsgFileAdd. Slave uzol využíva triedu TransQueueManipulateSlave pre obsluhu príkazov master uzla. Na príkazy slave uzol odpovedá správou MsgFileAck, ktorá obsahuje odhadovaný čas po vykonaní operácie a výsledok operácie. Výsledok má význam iba pri odoberaní súboru z fronty, keď je možné, že súbor už vo fronte nieje a je spracovávaný, prípadne spracovaný v minulosti. Master uzol postupuje nasledovne.

1. Vyhľadá súbor vo fronte uzlu uloženej v objekte NodeInfo reprezentujúci uzol. Súbor musí mať viacnásobný prístup a odhadovaný čas ukončenia výkonnejšieho uzlu musí byť menší aspoň o hodinu a 30 minút.
2. Pokúsi sa odstrániť súbor z fronty uzlu buď správou MsgFileRemove, alebo vyvolaním metódy removeFile lokálneho objektu typu TaskManager (prípade, keď pomalý uzol je samotný master).
3. V prípade, že sa podarí odstrániť súbor – MsgFileAck obsahuje true alebo removeFile() vráti true, tak sa pokúsi pridať súbor do fronty rýchlejšieho uzlu. V opačnom prípade pokračuje bodom 1. V každom prípade ale odstráni súbor z kolekcie reprezentujúcu frontu uzlu v objekte NodeInfo.
4. Súbor bude pridaný do fronty rýchlejšieho uzlu buď správou MsgFileAdd, alebo metódou addFile, ak sa jedná o master uzol. Ak sa jedná o slave uzol, postup je nasledujúci.
   - Ak sa potvrdí pridanie súboru správou MsgFileAck, je odoslaná správa MsgAck pre ukončenie transakcie. Ďalej je vyžadované potvrdenie ukončenia správou MsgAck. V prípade úspechu master pokračuje bodom 1.
   - Ak uzol neodpovedá, súbor je vrátený pomalému uzlu správou MsgFileAdd.

Pri pridávaní súboru do fronty je výsledok operácie pridania vždy true. Na obr. 4.5 je ilustrovaný presun dvoch súborov dvom rozličným rýchlejším uzlom.



Obrázok 4.5: Presun súboru s viacnásobným prístupom

Ak sa pri odosielaní správy MsgFileAck po požiadavke o odobratí súboru vyskytne výnimka IOException, súbor je pridaný späť do fronty a násilne sa ukončí transakcia.

Vyrovnanie záťaže medzi dvomi uzlami

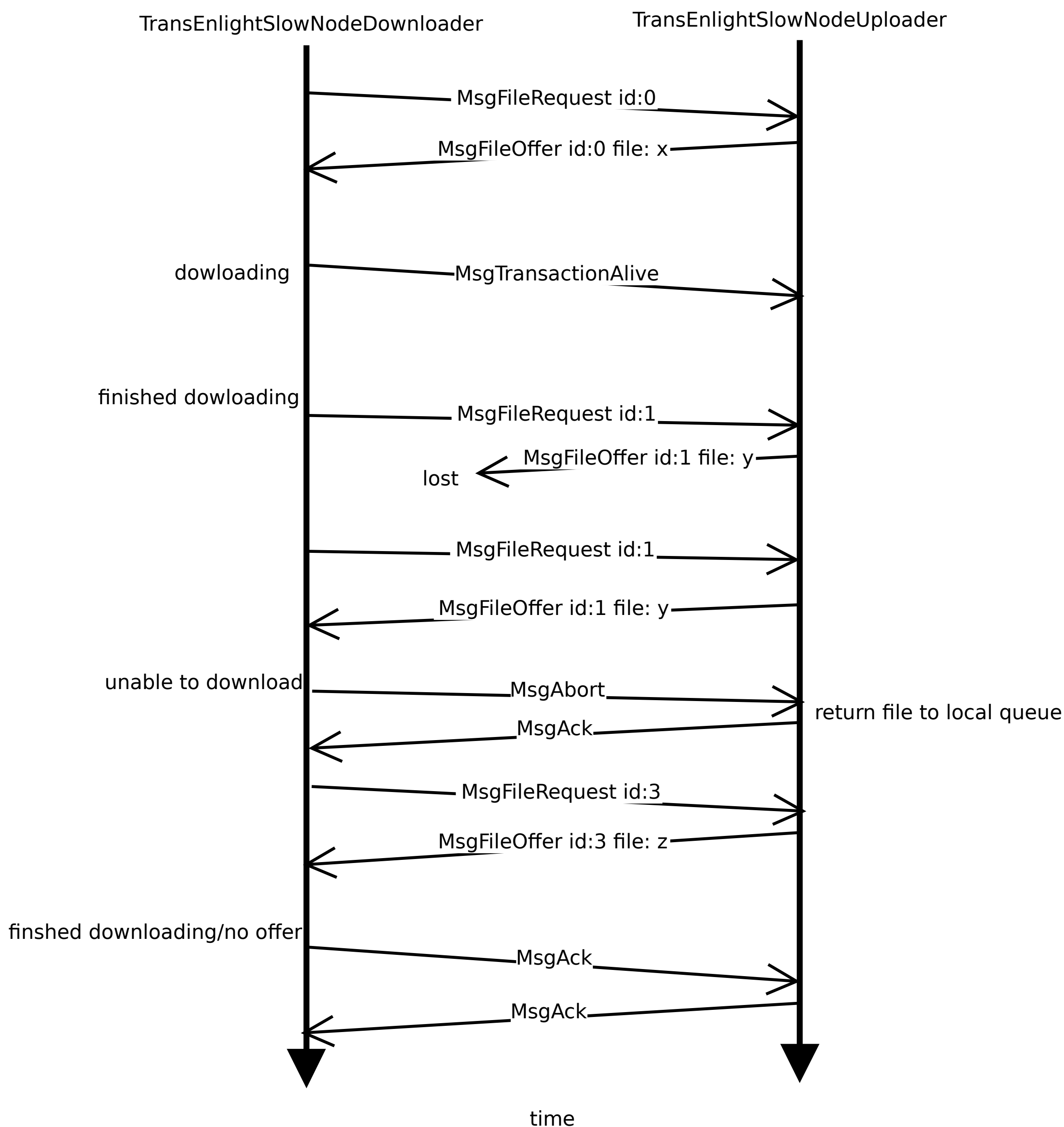
Ak master rozhodne o vyvážení záťaže medzi dvomi uzlami a uzol, ktorý má byť odľahčujúci je samotný master, vytvorí transakciu TransEnlightSlowNodeDownloader. V opačnom prípade vytvorí transakciu TransEnlightSlowNodeMasterCommand, ktorá má za úlohu prikázať príslušnému uzlu vytvoriť danú transakciu. Informuje uzol o FTP adrese a adrese, na ktorej načúva pomalý uzol. Postup odľahčovania je nasledujúci.

1. Rýchly uzol pošle požiadavok na súbor pomalému uzlu - MsgFileRequest. Pomalý uzol odoberie súbor z fronty a odošle správu MsgFileOffer, ktorá obsahuje odhadovaný čas ukončenia po odobratí a ak je vyžadovaný návrat výsledku, tak aj ftp adresu (NetAddress),

kam má byť výsledný súbor nahraný. Ak nieje žiadny súbor k dispozícii, prípadne získaný súbor už bol navrátený späť, je odoslaný prázdny reťazec miesto názvu súboru.

2. Ak rýchly uzol obdržal správu s neprázdnym reťazcom názvu súboru, začne súbor sťahovať pomocou triedy FileDownload. V opačnom prípade ukončí transakciu – bod 6.
3. Počas sťahovania súboru, každých 60 sekúnd odosiela správu MsgTransactionAlive pomalému uzlu, pre udržanie spojenia. V prípade ak pomalý uzol neobdrží akúkoľvek správu viac ako 10 minút, vrát posledný ponúknutý súbor späť do fronty a násilne ukončí transakciu.
4. Ak súbor bol stiahnutý v poriadku, tranksakcia informuje lokálny uzol typu Node o získaní súboru, ktorý následne informuje uzol master o získaní súboru a prípadne naplánuje vrátenie výsledku na získanú adresu. Ak súbor nebol získaný v poriadku, potom:
    1. Zašle správu MsgAbort a vyčká na správu MsgAck.
    2. Pomalý uzol vráti posledný ponúknutý súbor späť do fronty a prvý takýto súbor zaznamená. Ak neskôr získa súbor opäť z fronty, informuje rýchly uzol o prázdnej fronte – prázdny reťazec ako názov súboru. Ten transakciu ukončí – bod 6.
    3. Ak sa jedná o vážnu chybu, napríklad chybu spojenia, ukončí transakciu – bod 6.
5. Rýchly uzol porovná vlastný odhadovaný čas ukončenia so získaným, a ak je menší ako hodina a 30 minút, pokračuje bodom 1.
6. Rýchly uzol vykoná tento bod, ak sa rozhodne ukončiť transakciu. Pošle pomalému uzlu správu MsgAck značiacu koniec transakcie a vyčká na správu MsgAck. Následne informuje master-a o stave skončenia. Buď odľahčenie prebehlo v poriadku alebo pomalý uzol má prázdnu frontu, alebo vážna ftp chyba.

Správy MsgFileRequest a MsgFileOffer obsahujú poradové číslo – id. Na začiatku má hodnotu 0. Ak prídu dve správy MsgFileRequest s rovnakým id (dôvod môže byť nepridelenie procesorového času a následné nereagovanie pomalého uzlu), tak na každú správu odpovie rovnakým názvom súboru. Obe strany si uchovávajú posledné id. Ak rýchly uzol požaduje nový súbor, zvýši id o jedna. Ak teda pomalý uzol obdrží id o jedna väčšie ako posledné, vyberie z fronty nový súbor. Ilustrácia odľahčenia je na obr. 4.6.



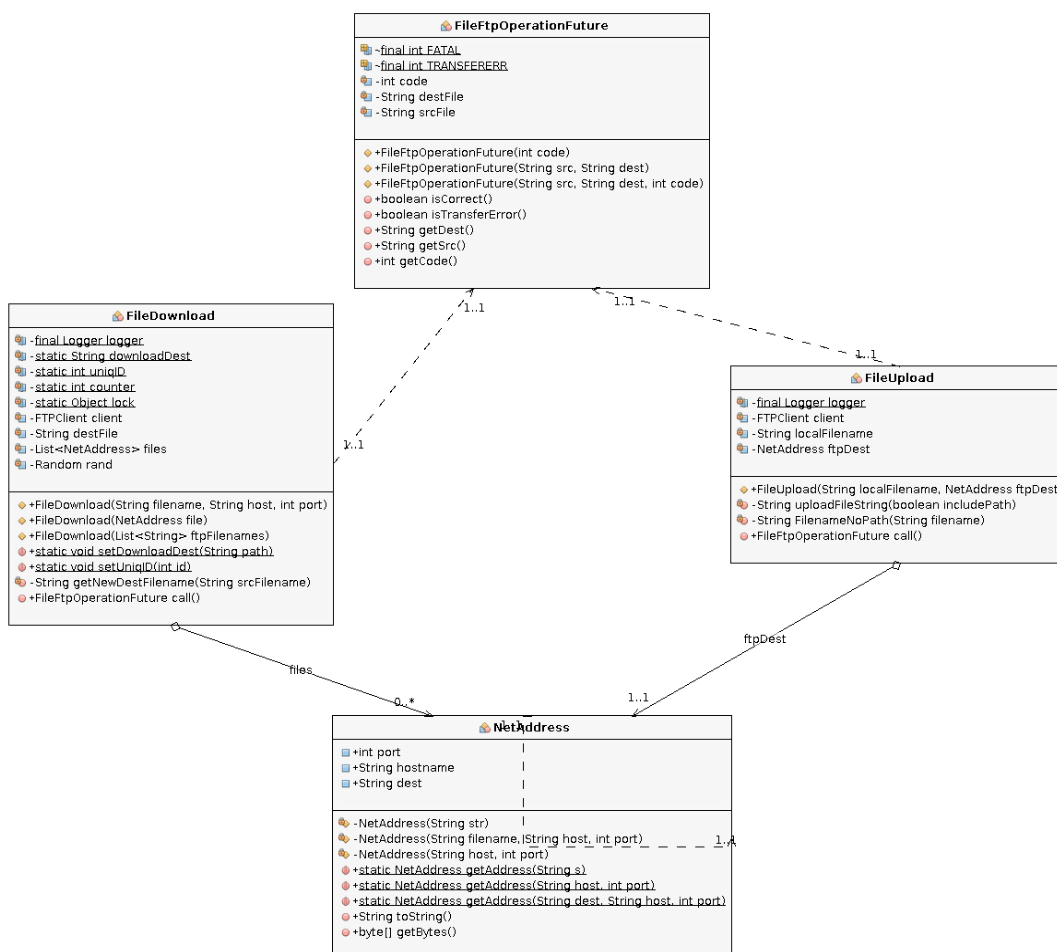
Obrázok 4.6: Vyvažovanie záťaže medzi dvoma uzlami

Ak sa na strane pomalého uzlu vyskytne výnimka IOException pri odosielaní správy, posledný ponúknutý súbor je vrátený späť do fronty a násilne sa ukončí transakcia.

Počas prenosu súboru je systém v nekonzistentnom stave. Ak nenastane chyba v sieti, tak po dokončení operácie systém bude v konzistentnom stave. Detekciu poruchy siete zabezpečuje vyžadovanie potvrdzovacích správ a pravidelné zasielanie správy MsgTransactionAlive počas sťahovania.

4.2.2.5 Prenos súborov

Pre sťahovanie a nahrávanie súborov je využívaná inštancia triedy FileDownload a FileUpload, ktoré implementujú rozhranie Callable a sú mienené, aby bežali konkurentne. Diagram tried je znázornený na obr. 4.7. Pre prenos je využitý FTP protokol.



Obrázok 4.7: Diagram tried pre prenos súborov

Konštruktor triedy FileUpload vyžaduje zdrojový súbor a ftp adresu zložky. Pri spustení, zavolanie metódy call, súbor nahraný do zložky so zmeneným názvom - pridaný prefix _upload_x_, pričom x je jednoznačné identifikačné číslo (ďalej id) uzlu.

Konštruktor triedy FileDownload akceptuje jednu ftp adresu alebo jej kolekciu v prípade, že je možné súbor stiahnuť z viacerých zdrojov. Ak je zdrojov viac, sú vyberané náhodne, v prípade, ak pokusy zlyhajú (chyba prenosu). Stiahnutý súbor má zmenený názov o prefix pd_corp_x_y_. X je číslo uzla a y je poradie stiahnutého súboru. Týmto mechanizmom sa predchádza možnému konfliktu pri rovnakých názvoch stiahnutých súborov.

Metódu call obidvoch objektov vracajú inštanciu triedy FileFtpOperationFuture hodnotiacu stav posledného pokusu prenosu. V prípade FileUpload je pokus a zdrojový súbor iba jeden. Objekt obsahuje cestu zdrojového súboru, cieľového súboru, FTP doménové meno, FTP port a status. Status môže byť trojaký.

- Správny – úspešný prenos.
- FTP nezávažná chyba – chyba, ktorá nieje chybou siete. Príkladom môže byť neexistujúci zdrojový súbor. O túto chybu sa jedná ak je FTP kód operácie 550.
- Vážna chyba – chyba prenosu, siete.

4.3 Detektor zlyhania

Master dokáže identifikovať zlyhanie výpočtového uzla, ak správa MsgHello nedorazila za posledných 10 minút. Po detekovaní je užívateľ informovaný o zlyhaní uzla. Ak zlyhá master uzol, užívateľ to identifikuje objavením zrušeného procesu. Master uzol nekomunikuje v pravidelných intervaloch so slave uzlami, preto je zlyhanie master uzla nezistiteľné otrockými uzlami.

4.4 Spracovanie súborov na jednom výpočtovom uzle

4.4.1 Návrh

Súbor je spracovávaný užívateľom stanoveným spôsobom. Užívateľ určí program, ktorý má spracovať súbory. Program bude spustený aplikáciou a môže spracovávať jeden súbor, a aplikácia spustí niekoľko podprocesov konkurentne alebo program bude komunikovať s aplikáciou pomocou štandardného vstupu a výstupu a získa cestu k súboru, ktorý má spracovať. Aplikácia má znalosť, aký je odhadovaný čas spracovania ostávajúcich súborov. Cesty k súborom sú uložené v zdielanej FIFO fronte a príslušné zaobalujúce triedy sú konkurentne bezpečné. Trieda, delegátor, zabezpečujúca spracovanie súborov príslušným spôsobom musí umožňovať získanie odhadovaného času ukončenia, teoretického času ukončenia po pridaní alebo odobraní súboru, pridanie a odobranie súboru z fronty, nastavenie príkazu, odštartovanie , ukončenie, násilné ukončenie a poskytnutie informácie, či sú všetky súbory spracované, musí umožňovať informovať informovanie robotníkom o spracovaní nejakého súboru. Riadiaci uzol je otrocký uzol alebo koordinátor, ktorý sa zúčastňuje spracovávania

súborov. Riadiaci uzol implementuje rozhranie, pomocou ktorého je ho možné informovať o spracovaní konkrétneho súboru a

4.4.1.1 Program spracúvajúci jeden súbor

Delegátor vytvorí množinu podprocesov (ďalej robotníkov), ktorý odoberajú jednotlivé súbory z fronty a spúšťajú systémový proces, ktorý súbor spracuje. Robotník najprv získa z fronty súbor a vytvorí reťazec pre spustenie systémového procesu nasledovne. V užívateľom zadanom reťazci určujúci príkaz pre systémový proces bude každý podreťazec @file nahradený cestou k súboru a podreťazec @filename nahradený názvom súboru bez cesty. Príkaz 'cat @file | wc -l > /out/@filename.out' spočíta počet riadkov v súbore a výsledok uloží do zložky /out a názvu pôvodného súboru s koncovkou '.out'. Vytvorí sa príkaz pre systémový proces a následne sa spustí. Robotník čaká na jeho dokončenie a informuje delegátora o dokončenej úlohe. Tiež informuje lokálny riadiaci uzol o spracovaní súboru a ceste k výslednému súboru, ak ju je možné zistiť. Užívateľ definuje v konfigurácii reťazec, pomocou ktorého je možné identifikovať výsledný súbor. V uvedenom príklade by to bol '/out/@filename.out'. Keďže koncept nedovoľuje žiadnu komunikáciu so systémovým procesom, nie je inak možné zistiť cestu k výsledky procesu. Informácia o lokácii výsledku je pre vytvorenie konfiguračného súboru s výslednými súbormi alebo navrátenie súboru na pôvodný výpočtový uzol, ak bol prenesený z dôvodu vyrovnania záťaže. Delegátor pri vyvolaní oboch akcií ukončenia násilne ukončí všetky podprocesy a systémové procesy.

4.4.1.2 Program spracúvajúci viac súborov

Delegátor spustí jeden systémový proces a komunikátor bežiaci konkurentne. Komunikátor má rolu prostredníka, ktorý komunikuje s procesom, reaguje na jeho žiadosti o nový súbor na spracovanie a informovanie o spracovanom súbore. Komunikátor vyvoláva príslušné metódy delegátora a riadiaceho uzlu. Protokol komunikácie je nasledovný. Systémový proces zahajuje akciu, tak, že pošle správu komunikátorovi na svoj štandardný výstup, na ktorom komunikátor načúva a očakáva odpoveď na svojom štandardnom výstupe. Typy správ sú dva:

- Žiadanie o nový súbor na spracovanie – komunikátor odoberie súbor z fronty a pošle cestu k súboru systémovému procesu na štandardný vstup. Správa má formát '\n'.
- Informovanie o spracovaní súboru – proces na štandardný výstup napíše cestu spracovaného súboru a cestu k výslednému súboru oddelené tabulátorom. Ak je výsledných súborov viac, sú oddelené medzerami. Komunikátor informuje delegátora o spracovaní súboru a riadiaceho uzla o spracovaní súboru a jeho výsledku alebo výsledkoch.

Ak proces obdrží na štandardný vstup reťazec 'exit\n', tak ho komunikátor informuje, že žiadny ďalší súbor na spracovanie nie je k dispozícii a proces môže skončiť.

Delegátor pri vyvolaní akcie ukončenia zavolá metódu komunikátora, ktorá pošle reťazec 'exit\n' systémovému procesu. Následné čaká na ukončenie systémového procesu a ukončí komunikátora. Pri násilnom ukončení postupuje rovnako ako v prechádzajúcom popise s rozdielom, že nečaká na ukončenie systémového procesu. Pri tejto metóde môže vzniknúť takzvaný zombie proces.

4.4.2 Popis implementácie

Rozhranie delegátora je popísané abstraktnou triedou TaskManager a čiastočnú implementáciu poskytuje Abstraktná trieda Feeder, ktorá je podtriedou triedy TaskManager. Táto trieda pracuje s LinkedBlockingQueue a zabezpečuje konkurentne bezpečný prístup. Implementuje funkcie pre neblokované pridanie, blokované odobratie súboru robotníkom – metóda obtainTask a neblokované odobratie súboru – metóda getFile a vymazanie konkrétneho súboru z fronty. Implementuje funkčnosť na získanie odhadovaného času ukončenia nasledovne. Počíta, koľko súborov bolo spracovaných – delegátor musí byť o tejto skutočnosti informovaný vyvolaním metódy taskDone. Udržuje informáciu o počte všetkých súborov. Pri metóde obrainTask neznižuje tento počet na rozdiel od metódy getFile, ktorá je určená pre riadiaci uzol. Trieda Feeder implementuje metódu start, ktorá musí byť rozširujúcou triedou volaná. Táto metóda uloží informáciu o čase začiatku. Odhadovaný čas ukončenia je vypočítaný ako priemerný čas potrebný na spracovanie jedného súboru (aktuálny čas behu / počet spracovaných súborov), vynásobený počtom ostávajúcich súborov (počet všetkých súborov – počet spracovaných). Trieda tiež implementuje teoretický odhad ukončenia tak, že ako parameter získa vektor definujúci zmenu, napr. -1 značiac odobratie jedného súboru z fronty. Tento vektor je jednoducho pridaný k počtu ostávajúcich súborov. Metódy pre odhad času ukončenia vracajú číslo typu double hovoriaci čas v sekundách. Ak nie je ani jeden súbor spracovaný, metóda vráti kladné nekonečno. Odhadovaný čas ukončenia je validný, ak je spracovaných viac ako 5% súborov. Ak táto podmienka nie je splnená, metóda vráti kladné nekonečno. Na metódu na určenie teoretického odhadu sa táto podmienka nevzťahuje.

4.4.2.1 SingleFileFeeder

Trieda rozširuje triedu Feeder a jej inštancia má rolu delegátora pre spracovanie súborov konkurentne systémovými procesmi, ktoré spracúvajú jeden súbor. Vytvorí pool typu ExecutorService s maximálnym počtom vlákien rovný 1,5 násobku počtu dostupných vlákien pre aplikáciu. Implementuje metódy pre získanie reťazca pre spustenie systémového súboru a získanie cesty

k výstupného súboru. Obe metódy sú závislé na parametri, ktorý je cesta k vstupnému súboru získaného metódou obtainTask. Metóda start vyvolá metódu rodičovskú metódu start triedy Feeder a vytvorí robotníkov typu SingleFileExecutor, ktorí v cykle získavajú súbory na spracovanie, vytvárajú proces, ktorý ich spracuje a následne čakajú na dokončenie.

4.4.2.2 PipeFeeder

Trieda rozširuje triedu Feeder a jej inštancia má rolu delegátora pre spracovanie súborov jedným systémovým procesom. Delegátor tento proces vytvorí spolu s komunikátorom typu Listener, ktorý beží konkurentne. Komunikátor reaguje na správy od systémového procesu. V cykle blokujúco čaká na správu, po prijatí ju dekóduje a vyvolá príslušnú reakciu. Ak obdržal '\n', získa ďalší súbor z fronty pomocou metódy obtainTask a pošle ju na štandardný vstup procesu. Ak obdrží iný reťazec, ktorý má formát 'vstupný súbor \t výstupný súbor(ak viac, oddelené medzerou)', tak informuje riadiaci uzol o spracovanom súbore a výsledných súboroch. Delegátora informuje o spracovaní súboru v každom prípade.

4.5 Návrh a spôsob implementácie prenosu správ v sieti

Výpočtový uzol prijíma správy na jednom UDP porte stanoveným užívateľom. Uzlom je možné zasielať správy pomocou inštancií triedy NodeDevice, ktoré poskytujú metódu pre odosielanie správ. Objekt typu NetworkService modeluje vrstvu pre príjem a odosielanie správ ostatným uzlom. Zasielanie správ uzlu prebieha pomocou objektu NodeDevice, vytvorenými objektom NetworkService. Inštancie tried NodeDevice odosielajú správy na konkrétnu adresu pomocou objektu DatagramChannel, ktorý je pre výpočtový uzol len jeden. Metóda createDevice vráti objekt NodeDevice pre zadanú adresu a port. Neblokujúca metóda receive objektu NetworkService vráti inštanciu triedy ReceivedObject, ktorý obsahuje prijatú správu a objekt typu NodeDevice, pomocou ktorého je možné odpovedať. V prípade, ak nie je prijatá žiadna správa, vráti hodnotu null. Správa je typu Message.

4.5.1.1 Správy

Správa je inštancia triedy, ktorá je podtriedou triedy Message. Trieda Message vyžaduje od podtried implementovanie metód pre získanie objektu ByteArray, pre serializáciu o maximálnej dĺžke 1024 bajtov. Ďalej sú vyžadované metódy pre získanie dvoch identifikačných čísel nodeID a transactionID, pomocou ktorých je možné jednoznačne identifikovať operáciu, ktorá je vykonávaná dvomi

spolupracujúcimi uzlami. Od podtried je tiež vyžadované, aby mali verejnú premennú opCode, ktorá po serializácii (metóda getBytes) bude ako prvý bajt v poli bajtov. Je teda v rozsahu 0-255. Pomocou tejto premennej bude inštanciovaná príslušná trieda metódou createMessageFromPacket, ktorú poskytuje trieda Message. Trieda Message poskytuje metódy pre zápis a čítanie objektu String z rôznych stream-ov. Reťazce sú oddelené znakom '\0'.

Správy sú používané vyššími vrstvami pre komunikáciu a sú vytvárané abstraktnou triedou Message po prijatí paketu.

4.5.1.2 Serializácia a deserializácia správ

Každá správa musí implementovať metódu getBytes(), ktorá serializuje samotný objekt. Prvým bajtom je číslo reprezentované triednou premennou opCode. Deserializácia je implementovaná triednou metódou getMessage triedy Message. Podľa prvého bajtu (unikátny opCode v rámci množiny správ) v príchodzom pakete sa rozhodne, ktorá trieda bude inštanciovaná. Príslušná trieda musí poskytovať konštruktor inicializujúci objekt z prijatého paketu vytvoreným metódou getBytes inštanciou rovnakej triedy.

4.6 Konfigurácia

Užívateľ určuje konfiguráciu pomocou argumentov spúšťaného programu a konfiguračného súboru. Argumenty sú nasledovné.

- -m / --master : musí byť uvedený ako prvý argument, povinný
- -c / --config súbor: cesta ku konfiguračnému súboru, povinný*
- -k / --known_hosts súbor: cesta k súboru určujúci kľúče známych počítačov, nepovinný
- -r / --rsa súbor: cesta k privátnemu kľúču pre autentifikáciu, nepovinný
- -e / --return: nastaví nahratie výsledku spracovania, ak zdrojový súbor bol stiahnutý z iného uzlu, nepovinný.
- -n / --new_config súbor: vytvorenie základného konfiguračného súboru z množiny výsledkov a známych uzlov, nepovinný.
- -s / --checksum: kontrola totožnosti súborov s viacnásobným prístupom. Využitá hash funkcia md5. Nepovinný.
- -g / --generate vstup výstup: generovanie základného konfiguračného súboru zo súborov v zložkách na jednotlivých počítačoch definovanými vo vstupnom súbore. Výsledok bude v súbore výstup. Povinný*.

* - buď -g, alebo -c musí byť použitý, ale nie súčasne.

4.6.1 Forma konfiguračného súboru prepínača -g

Každý riadok definuje zložku, doménové meno počítača, užívateľské meno a ssh port. Súbory nachádzajúce sa v zložke budú získané a uvedené vo výslednom konfiguračnom súbore. Formát riadku je (bez bielych znakov): 'užívateľské meno@doménové meno:ssh port <tabulátor> zložka'

4.6.2 Forma konfiguračného súboru prepínača -c

Dokument má formát xml a koreňový uzol root. Jeho detské uzle definujú buď súbor alebo výpočtový uzol.

4.6.2.1 Konfigurácia výpočtového uzlu

Výpočtový uzol je definovaný xml uzlom node. Atribút type určuje typ výpočtového uzla či sa jedná o master výpočtový uzol, alebo slave výpočtový uzol. Ak je atribút 'master', potom je ďalej možné definovať atribút compute na hodnotu 'true' alebo 'false' v závislosti, či master uzol má tiež spracovávať súbory. Druhou možnou hodnotou atribútu type je 'slave' hovoriaci, že výpočtový uzol je typu slave. Výpočtový uzol slave sa vždy zúčastňuje spracovania súborov. Detské uzle ďalej určujú ďalšie vlastnosti.

- hostname: textový element určuje doménové meno, prípade ip adresa
- user: textový element určuje užívateľské meno pre ssh spojenie
- port s atribútom service: service môže nadobúdať hodnoty 'ssh', 'ftp', 'node', textový element definuje port danej služby
- logFile: textový element určuje súbor, kde sú uchovávané informácie o udalostiach na výpočtovom uzle
- downloadFolder: textový element určuje zložku, kam budú uložené stiahnuté súbory z iných výpočtových uzlov
- jarPath: textový element určuje cestu k spúšťanému binárnemu súboru
- task: textový element určuje proces, ktorý má byť spustený pre spracovanie súborov. Atribúty uzlu sú:
  - fileFeed: 'single' alebo 'all'. V prvej možnosti je použitá trieda SingleFileFeeder ako delegátor, v druhej PipeFeeder.
  - result: reťazec, pomocou ktorého je možné nájsť výsledný súbor. Príkladom môže byť '/out/@filename', kde '@filename' bude nahradený názvom spracovaného súboru. Tento spôsob je využitý, ak je atribút fileFeed nastavený na 'single'.

- returnFolder: zložka, kam majú byť nahrané výsledné súbory v prípade, že boli spracované na inom stroji z dôvodu vyváženia záťaže

Príklad:

```
<node type="slave">
	<hostname>pcknot.fit.vutbr.cz</hostname>
	<user>xstrak25</user>
	<port service="ssh">22</port>
	<port service="ftp">35468</port>
	<port service="node">12345</port>
	<downloadFolder>/download</downloadFolder>
	<jarPath>/binary.jar</jarPath>
	<logFile>/log </logFile>
	<task
		result="/out/@filename.out"
		returnFolder="/return"
		fileFeed="single">
			java -jar /processing.jar @file /out/
	</task>
</node>
```

4.6.2.2 Konfigurácia súboru

Súbor je definovaný uzlom file a detskými uzlami access určujúci výpočtový uzol, ktorý má k súboru lokálny prístup. Príklad:

```
<file type="parallel">
	<access host="uzol1.cz" path="/1" />
	<access host="uzol2.cz" path="/2" />
</file>
```

Atribút host určuje nakonfigurovaný výpočtový uzol podľa hostname. Atribút path lokálnu cestu.

5 Experimenty

5.1 Spracovanie dataset CommonCrawl

Spracovali sa warc záznamy CC-2014-49 na počítačoch pcknot, pcotrusina a pcdytrych. Experiment bol vykonaný dvakrát. Prvýkrát sa experiment uskutočnil dňa 24.4.2015, pričom bolo možné pozorovať veľkú záťaž stroja pcotrusina neznámym procesom. Záťaž je možné pozorovať v log súboroch /mnt/data/commoncrawl/xstrak25/log_24042015CC201449 na jednotlivých strojoch. FTP server nedostával dostatok procesorového času a prenos zlyhával. Riadiaci proces tiež niekedy nedostával dostatok procesorového času a odozva na správu trvala až 30 sekúnd. Je nutné uviesť, že po prvom experimente boli zmenené názvy niektorých tried, čo sa odzrkadlilo v log súboroch. Druhýkrát bolo spracovanie bez takejto záťaže.

	počiatočné rozdelenie		**spracované pre odľahčenie**		**spracované celkom**	
	GB	počet	GB	počet	GB	počet
pcknot	887	962	0	0	583	632
pcotrusina	331	360	116	125	439	477*
pcdytrych	740	792	197	213	936	1005
suma	1958	2114	313	338	1958	2014
čas	42.6 h					

Tabuľka 1: Spracovanie CC-2014-49, dátum:24.4.2015

*- z dôvodu chyby boli dva súbory spracované dvakrát (do tabuľky to premietnuté nie je). Chyba nastala, keď FTP server nemal dostatok procesorového času a prenos zlyhal. Odľahčujúci uzol úspešne vrátil súbor späť do fronty pomalého uzlu, ale neukončil riadne transakciu výmenou správ MsgAck. Master uzol po informovaní o chybe v prenose sa pokúsil pôvodnú transakciu na pomalom uzle príkazom zrušiť. Keďže tá nebola riadne ukončená, vyvolala sa metóda cancel a súbor bol pridaný do fronty znovu. Chyba však nemala značný vplyv na celkový čas.

	počiatočné rozdelenie		**spracované pre odľahčenie**		**spracované celkom**	
	GB	počet	GB	počet	GB	počet
pcknot	887	962	0	0	552	599
pcotrusina	331	360	184	199	492	535
pcdytrych	740	792	183	198	913	980
suma	1958**	2114	367	397	1957**	2114
čas	40.6 h					

Tabuľka 2: Spracovanie CC-2014-49, dátum: 9.5.2015

** - odlišnosť je zapríčinená chybou zaokrúhľovania programom du pre zistenie veľkosti, keďže celkový počet spracovaných súborov je rovnaký, čo je najpodstatnejšia informácia.

Súbory boli spracovávané programom /mnt/data/commoncrawl/NLP-slave-0.0.1-SNAPSHOT-jar-with-dependencies.jar na každom počítači.

5.2 Spracovanie súborov na výkonnejších strojoch

Na počítačoch athena2, athena4 a athena6 bolo vybraných niekoľko súborov z dataset-u CommonCrawl. Cieľom experimentu bol test aplikácie na výkonnejších počítačoch. Počítače sú vybavené dvomi procesormi Intel® Xeon® E5-2630 , pričom každé jadro je taktované na frekvenciu 2.60GHz. Počet dostupných vlákien je 24 a veľkosť cache pamäte je 15MB.

	plán		**spracované pre odľahčenie**		**spracované celkom**	
	GB	počet	GB	počet	GB	počet
athena2	59	60	0	0	27	27
athena4	20	20	16	16	36	36
athena6	20	20	17	17	37	37
suma	99*	100	33	33	100*	100
čas	2.9 h					

Tabuľka 3: Spracovanie dát na athena2, athena4, athena6

* - odlišnosť je s veľkou pravdepodobnosťou zapríčinená chybou zaokrúhľovania programom du pre zistenie veľkosti, keďže počet spracovaných súborov je rovnaký, čo je najpodstatnejšia informácia.

Experiment bol vykonaný dvakrát, prvý pokus zlyhal kvôli vyššej záťaži počítača athena2 a nemožnosti alokovania dostatočnej pamäte. Tabuľka 3 reprezentuje výsledky z 2. pokusu, keď bol počet vlákien, v ktorých sa majú súbory spracovávať znížený na jednonásobok počtu dostupných vlákien. S touto zmenou boli vykonané všetky nasledujúce experimenty.

5.3 Porovnanie so systémom Hadoop

Ďalší experiment je porovnanie spracovávania súborov s aplikáciou MapReduce a systémom Hadoop.

5.3.1 Návrh a popis implementácie porovnávanej MapReduce aplikácie

Aplikácia má za úlohu vertikalizovať warc záznamy. Samotný algoritmus pre vertikalizáciu textu bol prevzatý z predchádzajúcich experimentov. Postup spracovania súboru je nasledovný. Inštancia triedy WarcRecordReader prečíta jeden záznam zo súboru a jeho obsah je zapísaný do pola bajtov spolu s poradovým číslom záznamu, URI a typom obsahu. Map funkcia vertikalizuje záznam a výsledok priradí rovnakému kľúču, aký bol získaný. Reduce funkcia . Inštancia triedy MultiFileOutput zapíše výsledok na HDFS.

5.3.2 Popis experimentu

Pre experiment bolo vytýčených približne 30GB dát, rozdelených na 10MB súbory, obsahujúcich časť webového obsahu. Dáta boli vertikalizované MapReduce aplikáciou v systéme Hadoop a aplikáciou na vertikalizáciu textu spustenou aplikáciou (ďalej pd_corp), ktorú popisuje táto práca. Aplikácia pd_corp využila 3 počítače: athena2, athena4 a athena6. Systém Hadoop mal k dispozícii iba jeden výpočtový stroj, a to athena2.

	plán		**spracované pre odľahčenie**		**spracované celkom**	
	GB	počet	GB	počet	GB	počet
athena2	10	1063	0	0	10	1063
athena4	10	1063	0	0	10	1063
athena6	9,9	1062	0	0	9,9	1062
suma	29,9	3188	0	0	29,9	3188
čas	27.15 min					

Tabuľka 4: Spracovanie 30GB dát na pc athena 2, 4, 6

Spracovanie MapReduce aplikáciou trvalo 150,5 minút. Je potrebné dodať, že práca so systémom Hadoop bola značne komplikovaná. Na skupine počítačov pcknot, pcotrusina a pcdytrych sa nepodarilo aplikácii prideliť potrebné zdroje a na skupine počítačov knot27, knot28, verifit4 a verifit5 nestačila pridelená veľkosť operačnej pamäte pre JVM. Práca so systémom bola technicky náročnejšia.

6 Záver

Hlavným cieľom práce bolo vytvoriť aplikáciu, ktorá rozdelí úlohu viacerým počítačom a následne podľa času ukončenia mení počítače, ktoré majú spracovať konkrétne súbory tak, aby bola množina súborov spracovaná čo najskôr. Experimenty ukázali, že tento cieľ bol splnený. Systém podľa odhadnutých časov ukončenia jednotlivých uzlov vyvažoval záťaž presúvaním súborov cez sieť pomocou FTP protokolu. Prenesením vznikol viacnásobný prístup a ak bol pôvodný odhadnutý čas nepresný a neskôr sa zmenil, súbory boli jednoducho presunuté medzi jednotlivými frontami úloh počítačov. Teda aj v prípade výskytu nadmerne zaťaženého počítača, nebol rozdiel v konečných časoch jednotlivých počítačov väčší ako hodina a tridsať minút.

Prvý experiment ukázal že v prípade zlyhaní, ktoré boli spôsobené vyššou záťažou, ostal systém v konzistentnom stave a všetky súbory boli spracované. Experiment ukázal malú chybu, keď v prípade zlyhania FTP prenosu pri snahe o zachovanie konzistencie bol súbor spracovaný dvakrát. Chyba bola odstránená.

Pri spracovávaní súborov aplikáciou v druhom experimente sa ukázalo, že počet vlákien, v ktorých sa majú súbory spracovávať, je príliš vysoký (1,5 násobok vlákien na procesore). Pri viac zaťaženom počítači (athena2) dochádzalo k nedostatku pamäte a nemožnosti alokácie, a súbory nebolo možné spracovávať. Pri menej zaťažených počítačoch sa tento problém tiež prejavil, ale nie v takom rozmere. Iba niektoré vlákna nemohli alokovať dostatočne veľkú pamäť. Výpočet na athena2 zlyhal úplne. Z tohto dôvodu bol počet vlákien znížený na rovnaký počet, aký je na procesore.

Tretí experiment ukázal, že jednoduchý návrh MapReduce aplikácie a jeho implementácia je značne pomalšia. Pre porovnanie je potrebné prihliadnuť na to, že pd_corp aplikácia mala k dispozícii trikrát vyšší výpočtový výkon z dôvodu využitia troch výpočtových uzlov. Hrubý odhad výpočtového času, ak by pd_corp aplikácia mala k dispozícii iba jeden výpočtový stroj, je 81,45 minút. Metóda bola jednoduchá – zostavenie rovnice nepriamej úmery $x/27.15 = 3/1$, pričom x je odhadovaný čas pri využití jedného výpočtového stroja. Výsledný čas je skoro polovičný ako čas spracovania systémom Hadoop. Systém Hadoop by bolo možné využiť vtedy, ak je možné vytvoriť návrh využívajúci masívnu paralelizáciu, ktorú systém ponúka. Ale aj v tomto prípade sa môže stať, že Hadoop nebude rýchlejší z dôvodu veľkej náročnosti systému. Vytvorenie návrhu a jeho implementácia tiež vyžaduje pokročilé technické znalosti a pokročilé znalosti samotného systému. Pokročilé znalosti taktiež vyžaduje správna konfigurácia systému. Pre veľkú efektivitu je potrebné prideliť dostatočne veľkú operačnú pamäť, čo môže znemožniť inú prácu.

7 Vízie a ďalšie rozšírenia

Rozšíriť aplikáciu je možné niekoľkými smermi:

Prvým je rozšírenie o voľbu koordinátora, keď po výpadku master uzla je zvolený nový koordinátor.

Druhým je reakcia na zistenie výskytu chyby na uzle a jeho znovuspustenie. S tým súvisí rozšírenie o informovaní uzlov, ktoré vstupné súbory boli spracované. Musia byť informované všetky, aby v prípade nového koordinátora mali všetky uzly rovnaké informácie o systéme. Bude sa musieť zaviesť nová podmienka do modelu konzistencie, a to, aby všetky uzly vedeli, ktorý uzol spracováva ktorý súbor.

Tretím rozšírením je automaticky regulovaný počet vlákien, v ktorých sa spracovávajú súbory vzhľadom na využité zdroje.

8 Literatúra

[1] Distribuované systemy. STORM. *Mizici.com* [online]. 8.10.2006 [cit. 2015-04-18]. Dostupné z: http://www.mizici.com/article.php?aid=67

[2] TANENBAUM, Andrew S a Maarten Van STEEN. *Distributed systems: principles and paradigms*. 2nd ed. Upper Saddle River: Pearson Education, 2007, xviii, 686 s. ISBN 01-323-9227-5.

[3] BONDI, André B. *Characteristics of Scalability and Their Impact on Performance: Proceedings of the second international workshop on Software and performance - WOSP 2000*. New York: Association for Computing Machinery, c2000, x, 226 p. ISBN 15-811-3195-X.

[4] Scalability: Measures. *Wikipedia* [online]. 2012, 28.3.2015 [cit. 2015-04-18]. Dostupné z: http://en.wikipedia.org/wiki/Scalability

[5] High-throughput computing. *Wikipedia* [online]. [cit. 2015-05-08]. Dostupné z: http://en.wikipedia.org/wiki/High-throughput_computing

[6] Distributed systems: for fun and profit. *Mikito Takada (mixu)* [online]. [cit. 2015-04-18]. Dostupné z: http://book.mixu.net/distsys/single-page.html

[7] JIA, Weijia a Wanlei ZHOU. *Distributed network systems: from concepts to implementations*. New York: Springer, 2005, xxvii, 513 s. ISBN 0387238395.

[8] MATOUŠEK, Petr. *Síťové aplikace a jejich architektura*. 1. vyd. Brno: VUTIUM, 2014, 396 s. : obr., grafy. ISBN 9788021437661.

[9] Systems architecture. *Wikipedia* [online]. 25.3.2015 [cit. 2015-04-18]. Dostupné z: http://en.wikipedia.org/wiki/Systems_architecture

[10] LAMPORT, Leslie. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM* [online]. vol. 21, issue 7, s. 558-565 [cit. 2015-05-02]. DOI: 10.1145/359545.359563. Dostupné z: http://portal.acm.org/citation.cfm?doid=359545.359563

[11] FIDGE, Colin J. *Timestamps in Message-Passing Systems That Preserve the Partial Ordering* [online]. Canberra: Department of Computer Science, Australian National University, február 1988 [cit. 22.4.2015]. Australian Computer Science Communications: 10, 1. Dostupné z: http://zoo.cs.yale.edu/classes/cs426/2012/lab/bib/fidge88timestamps.pdf

[12] KSHEMKALYANI, Ajay D a Mukesh SINGHAL. *Distributed computing: principles, algorithms, and systems*. Cambridge: Cambridge University Press, c2008, xvii, 736 s. ISBN 978-0-521-18984-2.

[13] *RFC 768: User Datagram Protocol* [online]. 28 August 1980 [cit. 23.4.2015]. Dostupné z: http://www.ietf.org/rfc/rfc768.txt

[14] INFORMATION SCIENCES, University of Southern 4676 Admiralty Marina del Rey, California 90291. *RFC: 793: TRANSMISSION CONTROL PROTOCOL* [online]. September 1981 [cit. 28.4.2015]. Dostupné z: https://www.ietf.org/rfc/rfc793.txt

[15] BROWNE, Julian. *Brewer's CAP Theorem: The kool aid Amazon and Ebay have been drinking* [online]. 11.01.2009 [cit. 27.04.2015]. Dostupné z: http://www.julianbrowne.com/article/viewer

[16] BREWER, Eric. *CAP Twelve Years Later: How the "Rules" Have Changed* [online]. InfoQ & IEEE Computer Society, 30.5.2012 [cit. 27.4.2015]. Dostupné z: http://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed

[17] JENKOV, Jakob. *Concurrency Models* [onlnie]. [cit. 27.4.2015]. Dostupné z: http://tutorials.jenkov.com/java-concurrency/concurrency-models.html

[18] WHITE, Tom. *Hadoop: the definitive guide*. 3rd ed. Sebastopol: O'Reilly, 2012, xxiii, 657 s. ISBN 978-1-449-31152-0.

[19] RICART, Glenn a Ashok K. AGRAWALA. An optimal algorithm for mutual exclusion in computer networks. *Communications of the ACM* [online]. 1981, vol. 24, issue 1, s. 9-17 [cit. 2015-05-02]. DOI: 10.1145/358527.358537. Dostupné z: http://portal.acm.org/citation.cfm?Doid=358527.358537

[20] LAMPORT. How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs. *IEEE Transactions on Computers* [online]. 1979, C-28, issue 9, s. 690-691 [cit. 2015-05-02]. DOI: 10.1109/TC.1979.1675439. Dostupné z: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1675439

Príloha 1

Zoznam použitých knižníc a programov

- Pre ssh bola použitá knižnica JSch[1]
- Pre FTP server bola použitá knižnica Apache FtpServer[2]
- Pre FTP klient bola použitá knižnica ftp4j[3]
- Ďalej bola použitá knižnica slf4j[4] pre správny chod FTP knižnice a knižnica Apache Commons[5]
- V MapReduce aplikácii pre vertikalizáciu textu boli použité zdrojové kódy /mnt/minerva1/nlp/projects/corpora_processing na serveri minerva1.fit.vutbr.cz

1http://www.jcraft.com/jsch/

2http://mina.apache.org/ftpserver-project/

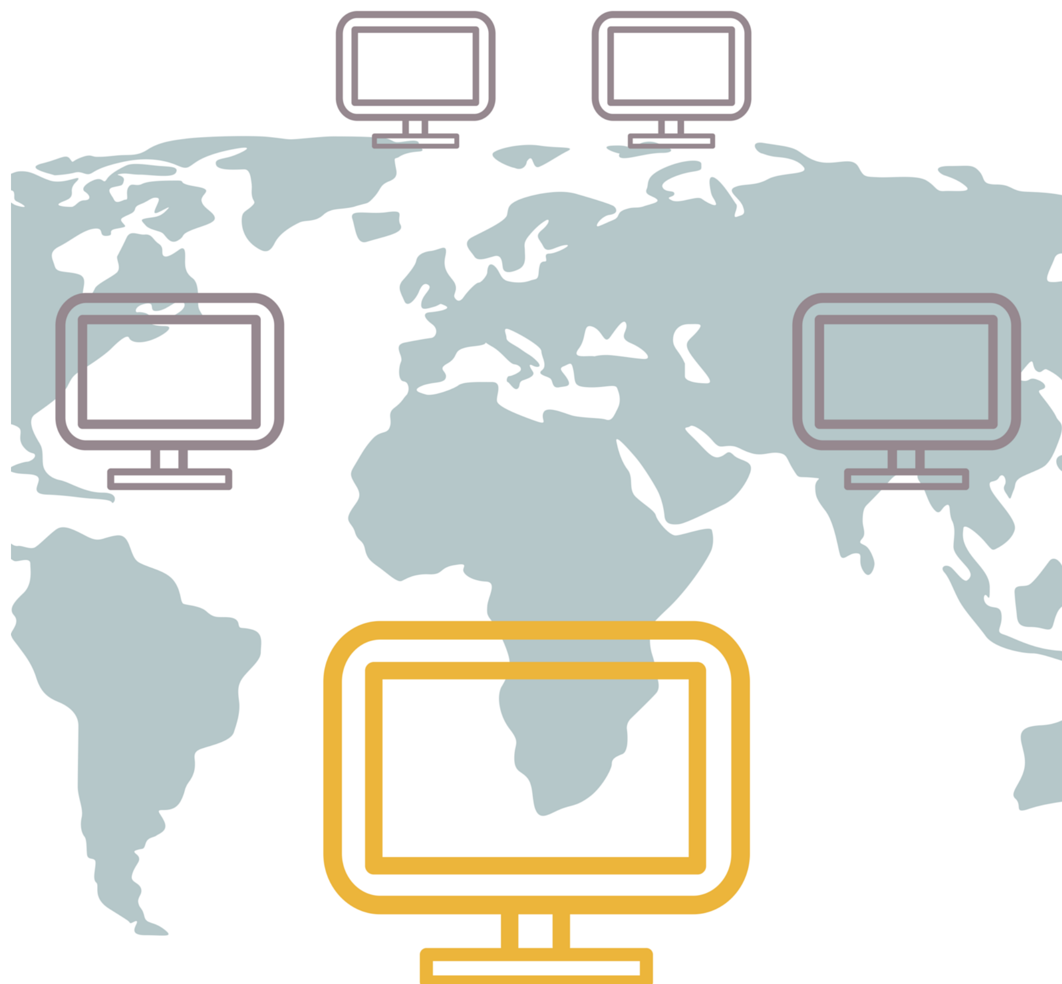
3http://www.sauronsoftware.it/projects/ftp4j/

4http://www.slf4j.org/

5http://commons.apache.org/proper/commons-io/

Príloha 2

Plagát



Lets be effective

together

as one

Zoznam príloh

Príloha 1. Zoznam použitých knižníc a programov.
Príloha 2. Plagát
Príloha 3. CD