Katedra Informatiky
Fakulta Matematiky, Fyziky a Informatiky
Univerzita Komenského, Bratislava

Distribuovaný zálohovací systém

(Bakalárska práca)

Marcel Ďuriš

Odbor: 9.2.1 Informatika

Vedúci: doc. RNDr. Rastislav Královič, PhD.

Bratislava, 2010

Čestne prehlasujem, že som túto bakalársku prácu vypracoval samostatne s použitím citovaných zdrojov.

.....

Ďakujem vedúcemu tejto bakalárskej práce doc. RNDr. Rastislavovi Královičovi, PhD. za ochotu pomôcť a poskytnúť svoje odborné názory a skúsenosti, bez ktorých by táto práca nedospela do svojej finálnej podoby. Ďakujem tiež bc. Richardovi Korenčiakovi za najrôznejšie druhy morálnej opory.

Abstrakt

Autor:	Marcel Ďuriš
Názov bakalárskej práce:	Distribuovaný zálohovací systém
Škola:	Univerzita Komenského v Bratislave
Fakulta:	Fakulta matematiky, fyziky a informatiky
Katedra:	Katedra informatiky
Vedúci bakalárskej práce:	doc. RNDr. Rastislav Královič, PhD.
Rozsah práce:	38 strán
Bratislava, jún 2010	

Táto bakalárska práca sa zaoberá návrhom, popisom a implementáciou zálohovacieho riešenia, ktoré využije diskové miesto bežných počítacov. Cieľom je využiť inak prázdny úložný priestor nespoľahlivých počítačov a navrhnúť systém, ktorý zabezbečí dostupnosť dát v prípade výpadku jednotlivých uzlov, ako aj rovnomerné rozloženie záťaže medzi týmito uzlami.

Kľúčové slová: záloha, Reed-Solomonov kód, informačná bezpečnosť

Abstract

Author:	Marcel Ďuriš
Thesis title:	Distributed backup system
University:	Comenius University, Bratislava
Faculty:	Faculty of Mathematics, Physics and Informatics
Department:	Department of Computer Science
Advisor:	doc. RNDr. Rastislav Královič, PhD.
Thesis length:	38 pages
Bratislava, june 2010	

This bachelor thesis deals with design, description and implementation of a backup solution which uses disk space of ordinary computers. Goal of this thesis is to make use of empty space of unreliable computers and design a system capable of maintaing data availability even in case of failure of several networked nodes along with load-balancing between these nodes.

Keywords: backup, Reed-Solomon code, information security

Predhovor

Záloha je asi najistejším spôsobom ako zaistiť bezpečnosť našich údajov. Zálohou môže byť súbor uložený na prenosnom médiu, v schránke doručenej pošty, ale aj bezpečne skrytý v ohňuvzdornom trezore chrániacom zálohovací server. Čo ale robiť, keď zlyhá záloha, keď sa pokazí zálohovací server, keď stratíme svoj prenosný disk? Urobiť zálohu zálohy?

Môžeme draho a prácne zálohovať zálohy, preposielať si poštu a kopírovať prenosné médiá. Môžeme poprosiť kamarátov, nech si nechajú pre nás nejaký kúsok miesta na ich disku. Nakoniec, keď príde tá chvíľa, môžeme sa stratiť v mori záloh, nevediac ktorá je tá pravá, tá ktorá nás má zachrániť.

Nedá sa to aj krajšie? Nebolo by pekné mať zálohu odolnú voči výpadkom vo vnútri nej samej? Keď už by sme našli takýto nástroj, čo všetko by nám umožnil? Bolo by to drahé alebo lacné, náročné na spravovanie alebo primitívne jednoduché? Boli kolegovia nervózni z pomaly sa kaziaceho serveru minulosťou?

Obsah

Zoznam obrázkov

Úvod

Cieľom tejto bakalárskej práce je implementovať systém pracujúci na viacerých počítačoch, ktorý by dokázal efektívne odolávať výpadkom či už počítačovej siete, alebo uzlov samotných. Je dôvod domnievať sa, že kancelárske počítače, aj keď nie sú príkladom najvýkonnejších počítačov, nie sú využívané naplno najmä čo sa týka úložného miesta. Využitie týchto počítačov však znamená mnoho problémov, ktoré budeme musieť riešiť.

Práca sa pokúsi uviesť čitateľa do problematiky zálohovania ale aj načrtnúť rôzne riešenia nášho problému a poskytnúť náhľad na výhody ale aj nevýhody s nimi spojené. Popísané budú Reed-Solomonove kódy, ktoré sa javia ako vhodné riešenie spomínaného problému, ale aj detaily spojené s ich efektívnym využitím na viacerých počítačoch.

Kapitola 1

Zálohovanie

V tejto kapitole sa pokúsime špecifikovať problém, ktorý hodláme riešiť. Najprv ale definujeme základné pojmy, ktoré budeme ďalej používať, a poskytneme náhľad do problematiky zálohovania. Popíšeme si životný cyklus súboru a jeho vzťah k informačnej bezpečnosti.

1.1 Dokument a súbor

Nebudeme ďaleko od pravdy, ak povieme, že informačné systémy sú vyvíjané na to, aby spracovávali údaje. Často sa jedná o veľké množstvo informácií a práve tie sú často najhodnotnejším komponentom systému. S údajmi sa môžeme stretnúť v nespočetnom množstve formátov a reprezentácií. Pod pojmom dokument budeme rozumieť ľubovoľný objekt, ktorý obsahuje údaje a používa sa na ich prenos, uchovávanie ale aj prezentáciu človeku. Dokumenty sú nutne zviazané na médium, ktoré ich uchováva. Príkladom média môže byť papier, ale aj veľké množstvo rôznych druhov elektronických médií. Na základe média môžeme intuitívne rozlišovať *klasické* alebo *elektronické* dokumenty.

> **Poznámka:** *Dokument nie je určený len médiom ale aj formátom a obsahom. Formát nám hovorí, ako dokument čítať alebo akým spôsobom ho správne zapisovať. Obsah je samotná informácia dokumentu - to čo mu dáva hodnotu.*

Ako súbor budeme v tejto práci chápať objekt uložený na elektronickom médiu, ktorý uchováva dokument alebo iné údaje. Súbor môže obsahovať okrem dokumentov aj iné informácie používané počítačom ako napríklad spustiteľný program.

Stručne si popíšme, ako vyzerá životný cyklus súboru.

Vytvorenie. Po získaní dostatočného množstva údajov im môžeme dať formu a uložiť ich. Vytvorili sme nový súbor.

Spracovanie a použitie. Na rozdiel od klasických dokumentov sa tie elektronické ľahšie upravujú. Súbory preto od vzniku môžu prejsť značnými zmenami.

Archivácia. Dôležité súbory má zmysel uchovávať aj dlho po tom, ako prestalo byť dôležité meniť ich obsah. Tie sú uložené do archívu, ktorý ich zachováva pre neskoršie použitie.

Zničenie. Ak pre nás prestal byť súbor dôležitý, môžeme ho zničiť. Získame tak zdroje, ktoré sme potrebovali na uchovávanie dokumentu výmenou za stratu jeho obsahu.

1.2 Archív a jeho bezpečnosť

Ako už bolo spomenuté, súbory je potrebné uchovávať na nejakom médiu. Archív je mienený ako primárne úložisko všetkých našich súborov[1] a nachádza svoje miesto pri všetkých fázach životného cyklu dokumentu. Na to, aby sme archív mohli s kľudným svedomím používať, kladieme na neho nasledujúce požiadavky:

Integrita. Nechceme aby sa údaje, ktoré v archíve uchovávame samovoľne menili. Požadujeme, aby medzi jednotlivými cielenými úpravami dokumentu bol jeho obsah nemenný. Ľahko si vieme predstaviť prípady, kedy by narušenie integrity údajov malo fatálne následky.

Dostupnosť. Archív stráca svoj význam, ak k údajom v ňom nemáme prístup.

Dôvernosť. Určité údaje majú význam nielen pre ich majiteľa, ale aj pre iných, ktorí môžu údaje zneužiť vo svoj prospech, prípadne ich použiť proti pôvodnému majiteľovi. Je dôležité, aby sa údaje nedostali do nesprávnych rúk.

Zabezpečenie neustálej platnosti týchto troch podmienok, a nielen ich, je odborom skúmania disciplíny zvanej *informačná bezpečnosť*. Na to aby sme náš archív, teda aj naše údaje, udržali v bezpečí sa teda potrebujeme chrániť nielen pred nepovolanými zásahmi do údajov, ale aj pred rôznymi katastrofickými scenármi. Informačná bezpečnosť aj napriek tomu, že si to nemusíme na prvý pohľad uvedomiť, má širší obor pôsobenia ako len počítače.

[1] Naša definícia nie je úplne v spore so zaužívaným chápaním archívu ako úložiska starých dokumentov. Naša definícia rozširuje archív o tie dokumenty, s ktorými sa stále pracuje.

Poznámka: *Príkladom použitia informačnej bezpečnosti v praxi môžu byť aj také samozrejmé veci ako je zamknutie dverí. Pripravení musíme byť nielen na záškodnícke úmysly, aj zásah prírodného živlu môže ovplyvniť funkčnosť archívu.*

1.3 Záloha

Žiaľ, nie všetkému sa dá predchádzať, niekedy musíme byť pripravení na nové okolnosti. Rozumným riešením sa zdá byť ďalšia kópia dokumentov, ktorá zabezpečí dostupnosť údajov v prípade nepoužiteľnosti archívu. Rozdiel medzi archívom a zálohou je v tom, že archív obsahuje primárnu kópiu dokumentov, s ktorými sa aj priamo pracuje. Avšak k dokumentom v zálohe pristupuje až vtedy, keď archív prestal plniť svoj účel.

1.3.1 Tvorba zálohy

Pred tým, ako začneme vytvárať zálohu nášho archívu, potrebujeme mať k dispozícií úložné médiá. Taktiež je žiadúce, aby zálohy na nich boli vytvárané s určitou mierou organizácie. Pre účely tejto práce budeme chápať depozitár ako množinu elektronických médií, na ktorých bude uložená záloha. Zálohu môžeme v depozitári vytvárať viacerými spôsobmi.

Neštruktúrovaná záloha. Najjednoduchší spôsob, ako vytvoriť zálohu je ich ručné kopírovanie na iné médium, napr. CD. To môžeme, ale nemusíme, poznačiť a odložiť k ostatným. Výhodou tohto riešenia je jednoduchosť, akou vznikajú nové zálohy. Nevýhodou je náročnosť obnovy.

Plná záloha V princípe jednoduchý, avšak nie vždy šetrný spôsob zálohovania dát je kopírovanie všetkých chránených súborov. Zrejmá výhoda je, že ich obnova je opäť jednoduché kopírovanie súborov, čiže to je relatívne rýchly proces. (Nie je potrebná tak zložitá réžia, ako u ostatných prístupov.) Nevýhodou je vysoká spotreba úložného miesta.

Diferenciálna záloha Zálohujú sa iba tie súbory, ktoré boli zmenené od poslednej plnej zálohy. Na obnovu sú potrebné dve zálohy, najaktuálnejšia plná a diferenciálna záloha. Tento prístup je kompromisom medzi plnou a inkrementálnou zálohou.

Inkrementálna záloha Zálohované sú súbory, ktoré boli zmenené od poslednej inkrementálnej alebo plnej zálohy. Tento prístup je najšetrnejší k úložnému miestu nako-

ľko sa zálohujú iba súbory, ktoré boli zmenené od poslednej zálohy. Na druhej strane si obnova takto vytvorenej zálohy vyžaduje najviac námahy, pretože je potrebná posledná plná záloha a všetky inkrementálne zálohy vytvorené od tejto plnej zálohy.

Stupňovaná záloha Tento prístup je akýmsi zovšeobecnením diferencialnej a inkrementálnej zálohy. Pred vytvorením zálohy jej priradíme číslo, jej stupeň. Následne zálohujme zmeny súborov, ktoré nastali od vykonania poslednej zálohy s menším stupňom.

Kontinuálna ochrana Údaje sú chránené priebežne a všetky zmeny sú zaznamenané. Výhodou tohto prístupu je možnosť obnovy ľubovoľnej verzie. Tento prístup tiež môže za určitých okolností efektívne využívať zálohovacie médium. Obdoba kontinuálnej ochrany sa používa v databázových systémoch na zaistenie trvácnosti vykonaných operácií.

1.3.2 Rotačné schémy

Neustálym vytváraním ďalších a ďalších záloh môže množstvo dát v našom repozitári neúnosne rásť. Naproti tomu staré zálohy môžu strácať na svojej dôležitosti. Je preto rozumné príliš staré zálohy zmazať a uvoľnené miesto použiť pre tie nové. Vyvstáva otázka, ako určiť, ktorú zálohu už nepotrebujeme. Opäť môžeme využiť viacero rôznych spôsobov[2].

Round Robin

Veľmi jednoduchý spôsob je cyklické rotovanie médií, vždy zmažeme najstaršiu zálohu a uvoľnené miesto použijeme na vytvorenie novej.

GFS - Grandfather Father Son

Tento prístup udržiava hierarchiu pri vytváraní záloh. Periodicky, napríklad denne, sa vytvárajú zálohy - synovia. Každých niekoľko záloh je posledný zo synov povýšený na otca, ktorý je skopírovaný na vlastné médium. Médiá synov sú ďalej cyklicky používané obdobne ako v prístupe Round Robin. Médiá otcov sú tiež používané cyklicky a tiež pravidelne posledného z otcov povýšime na starého otca.

[2]Mnohé z týchto spôsobov dosahujú lepšie výsledky s použitím vhodných štruktúr zálohy a s ich rozumným plánovaním v čase.

Médium / Deň	1	2	3	4	5	6	7	8	9
Syn	*	*		*	*		*	*	
Otec			*			*			
Starý otec									*

Obr. 1.1: Príklad algoritmu GFS - Syn je vytváraný každý deň, každý tretí syn je povýšený na otca, každý tretí otec je povýšený na starého otca

Médium / Deň	1	2	3	4	5	6	7	8
A	*		*		*		*	
B		*				*		
C				*				
D								*

Obr. 1.2: Príklad algoritmu hanojských veží - Použili sme štyri sety médií

Ako príklad nám poslúži nasledovný postup pri vytváraní a uchovávaní záloh. Nová záloha je vytvorená každý deň. S ohľadom na úložné miesto je to diferenciálna záloha[3]. Raz týždenne, povedzme v piatok, je vytvorená plná záloha, vzniká otec. Vždy v posledný piatok v mesiaci povýšime posledného otca na starého otca, ktorý je uchovávaný pol roka.

Výhodou tohto prístupu je, že máme síce obmedzený, ale predsa nejaký prístup k veľmi starým zálohám, starým otcom, ako aj k tým novším. Samozrejmosťou je prístup k najaktuálnejšej zálohe, ale aj tým niekoľko dní starým.

Hanojské veže

Zálohovacie média rotujeme obdobne ako hracie kamene pri riešení spomenutého hlavolamu. Rozdeľme médiá do niekoľkých skupín. Médium z prvej skupiny používame pri každej druhej zálohe. Médium z druhej skupiny pri každej štvrtej, začínajúc druhou zálohou. Médium z tretej skupiny nájde uplatnenie pri každej ôsmej zálohe. Ľahko vidieť, že čím staršie zálohy uchovávame, tým väčšie časové intervaly medzi nimi sú. Algoritmus Hanojských veží je síce komplikovaný v porovnaní s ostatnými, má však aj svoje výhody. Rekurzívny algoritmus umožňuje ľahké rozšírenie pre ďalšie zálohovacie médiá. Využitím tohto algoritmu získame zálohy z $1, 2 \ldots 2^n$ dní dozadu.

[3]Vidieť, ako vhodný prístup k tvorbe zálohy ovplyvní využitie médií.

1.3.3 Version control

Zálohy nemusia byť použité len v prípade zlyhania archívu. Uplatnenie nájdu aj pri bežnej práci, a to napríklad vtedy, ak používateľ urobil do súboru zásahy, ktoré chce neskôr zrušiť. Iným príkladom využitia záloh je sledovanie zmien, napríklad hľadania časti programového kódu, „ktorý ešte fungoval“. Hlavne pri spolupráci viacerých pracovníkov na jednom projekte budú takéto možnosti využité, avšak na skutočnú použiteľnosť je potrebný systém, ktorý dokáže správne spájať dokumenty súčasne upravované viacerými spolupracovníkmi.

1.4 Existujúce zálohovacie riešenia

V tejto kapitole sa zoznámime s existujúcimi systémami na vytváranie a správu záloh.

1.4.1 Voľne dostupné nástroje

Jednoduché nástroje

Túto skupinu tvoria prevažne unixové utility na správu súborov. Jednoduchosť týchto nástrojov a taktiež fakt, že sa dajú plne ovládať pomocou príkazového riadka, umožňuje ich automatizované použitie pomocou rôznych zálohovacích skriptov. Tieto skripty môžu systémoví administrátori vytvárať a upravovať podľa svojich potrieb. Výsledkom je jednoduchý softvér ušitý na mieru daným podmienkam. Na druhej strane majú tieto nástroje značné obmedzenia, nehovoriac o možnej chybovosti použitých skriptov.

Príkladom využitých nástrojov môžu byť unixové `cp`, `mv`, `tar`, `cpio`, `dump` ale aj `dd`. Využitie v skriptoch si nájdu aj iné programy ako `find`, `cron` či `ls` a `grep`.

Komplexné nástroje

Neduhmi jednoduchých nástrojov by už nemali trpieť profesionálne nástroje ako Amanda alebo Bacula. Tieto disponujú bohatou funkcionalitou ako napríklad vytváranie záloh po sieti a plánované zálohovanie. Softvér Amanda dokonca nie je potrebné inštalovať na zálohovaný počítač, zálohovací server totiž dokáže pristupovať k zálohovaným údajom protokolmi SAMBA alebo NFS.

Nevýhodou niektorých nástrojov z tejto skupiny môže byť komplexnosť ich konfigurácie, čo môže spôsobiť problémy ak majú byť použité menej technicky zdatným používateľom.

1.4.2 Komerčné riešenia

V tejto skupine sa dajú nájsť okrem neprehľadného množstva jednoduchých zálohovacích programov, často určených pre operačné systémy Windows, aj komplexné zálohovacie riešenia. Tieto komerčné nástroje sa vyznačujú svojou pestrosťou. Dajú sa nájsť také, ktoré nemajú ani funkcionalitu voľných nástrojov, ale aj také, ktoré disponujú širokou škálou možností ako sú simultánne zálohy, zálohy na nízkej úrovni, ba dokonca aj vzdialené zálohy, ktoré by sme asi len ťažko hľadali u voľného softvéru.

1.4.3 Zhrnutie

K dispozícií máme veľké množstvo zálohovacích produktov s pestrou funkcionalitou, ktorá hlavne pri komerčných riešeniach aj niečo stojí. V každom prípade si je z čoho vyberať.

1.5 Motivácia

Nech sa už rozhodneme pre komerčný alebo otvorený softvér, určite budeme potrebovať depozitár, do ktorého svoju zálohu uložíme. Avšak kde nájsť čo možno najväčšie množstvo úložného miesta za čo možno najmenšiu cenu?

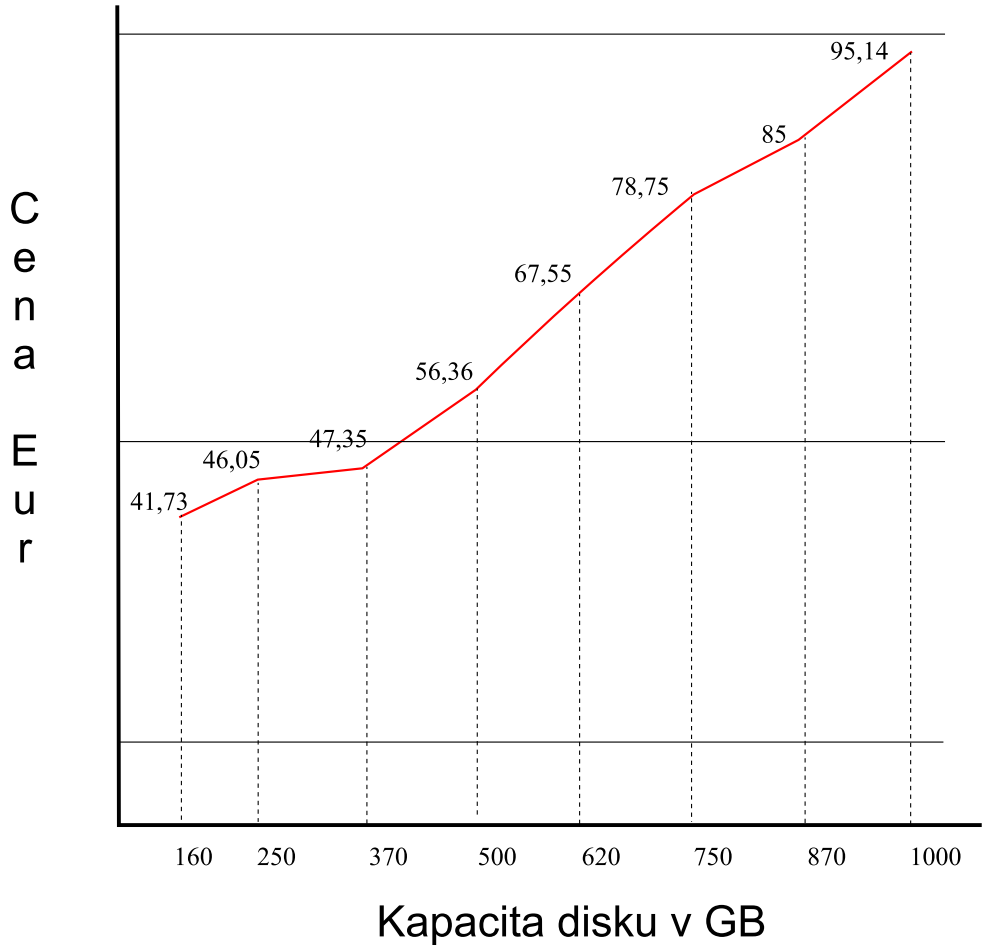
V dnešnej dobe si môžeme dovoliť kúpiť výkonný počítač aj na jednoduchú kancelársku prácu. Pri práci zahŕňajúcej evidovanie objednávok, spracovanie faktúr a písanie elektronickej pošty je málo pravdepodobné, že sa dlhodobo bude využívať výkon počítača naplno. Pravdepodobne aj úložné miesto pri určitom využití počítača ostane nevyužité. Naproti tomu si v dnešnej dobe len za malý príplatok môžeme kúpiť do relatívne väčší pevný disk. V niektorých prípadoch je možné dokonca kúpiť si o pár eur drahšie úložné médium s niekoľkonásobne väčšou kapacitou.

V poslednej dobe sa čoraz viac hovorí o virtualizácií, prečo sa teda nepokúsiť „virtualizovať“ inak nevyužité miesto na množstve kancelárskych počítačov a využiť ho napríklad na uchovanie zálohy, ktorá nám môže ušetriť nemalé náklady.

1.5.1 Dôsledky

Použitie bežných kancelárskych počítačov na tak zodpovednú úlohu ako je uchovávanie zálohy prináša problémy, ktoré musíme brať do úvahy. Jedným z nich je samotná spoľahlivosť takýchto počítačov. Dlhodobé výpadky spôsobené poruchami hardvéru a programového vybavenia alebo krádežou budú striedané reštartami a vypínaním počítača.

Aj keď nám kancelársky počítač popri svojej bežnej práci môže poskytnúť pridanú hodnotu vo forme lacného úložného miesta, je to za cenu nízkej spoľahlivosti, ktorú budeme musieť vyvážiť systémom odolným voči výpadkom.



Obr. 1.3: Graf znázorňujúci závislosť ceny pevného disku od jeho kapacity. Aj napriek tomu, že priemerná cena disku rastie lineárne, najmä pri menších diskoch vidno zreteľný prírastok kapacity za malú cenu. Údaje boli získané v decembri 2009 z internetových obchodov Alza, pocitac.sk a OfficeLand

Kapitola 2

Špecifikácia

Prvou fázou životného cyklu snáď každého projektu je určenie požiadaviek na výsledný softvér. Cieľom tejto kapitoly je bližšie špecifikovať funkcionalitu výslednej aplikácie.

2.1 Vytvorenie zálohy

Aplikácia poskytne užívateľovi možnosť vybrať si súbor, ktorý má byť zálohovaný. Zvoliť bude možné iba súbor, ktorý má užívateľ právo aspoň čítať, inak bude informovaný o nemožnosti vytvoriť zálohu. Každej vytvorenej zálohe bude priradená informácia, na základe ktorej ju bude možné jednoznačne identifikovať.

Zálohovaný súbor bude následne uložený na počítačoch v sieti tak, aby výpadok zhora ohraničeného počtu počítačov neovplyvnil dostupnosť zálohy.

2.2 Plánovanie zálohy

Užívateľ bude môcť k zvolenému súboru zadať dátum a čas. Na základe týchto údajov bude záloha pravidelne vytváraná. Užívateľ si bude môcť vybrať z vytvárania záloh každý deň, raz v týždni alebo raz v mesiaci.

2.3 Vyhľadanie zálohy

Používateľ bude mať možnosť zadať názov súboru, ktorý bol predtým zálohovaný. Aplikácia následne kontaktuje aplikačných klientov bežiacich na ostatných počítačoch v sieti aby tak mohla získať zoznam relevantných záloh a informácií o zálohovaných súboroch. Užíva-

teľovi bude následne zobrazený prehľadný zoznam záloh obsahujúci aspoň názov daného súboru, čas, v ktorom bol zálohovaný, a meno používateľa, ktorý zálohu vytvoril.

2.4 Obnovenie zálohy

Vytváranie záloh bez možnosti ich obnovy by nemalo zmysel, preto aplikácia umožní zvoliť zálohu zo zoznamu a následne ju obnoviť. Aplikácia vrámci procesu obnovy kontaktuje inštancie na ostatných počítačoch aby získala zoznam dostupných fragmentov pôvodného súboru a súborov obsahujúcich kontrolné súčty. Niektoré z nich budú následne stiahnuté na lokálny počítač a pôvodný súbor bude obnovený. Užívateľovi bude umožnené vybrať si, kam sa má obnovený súbor uložiť.

2.5 Odstránenie zálohy

Vrámci šetrného využívania zdrojov bude umožnené manuálne zmazať zálohu podľa uváženia používateľa. Aplikácia bude taktiež využívať vhodnú rotačnú schému aby sa tak predišlo zbytočnému ukladaniu príliš starých alebo nepotrebných záloh.

2.6 Grafické rozhranie

Grafické používateľské rozhranie musí byť jednoduché a prehľadné aby používateľ naučil ovládať aplikáciu v čo možno najkratšom čase. Rozhranie bude rozdelené do dvoch sekcií a to „Vytvorenie zálohy“ a „Vyhľadanie a obnova zálohy“. V aplikačnom menu bude môcť používateľ vyvolať konfiguračný dialóg aplikácie, vybrať súbor na zálohovanie, zobraziť informácie o samotnej aplikácií ale ju aj ukončiť. K najčastejšie vykonávaným činnostiam bude priradená klávesová skratka umožňujúca rýchle a jednoduché vykonanie danej činnosti.

Nakoľko bude aplikácia dlhodobo spustená, je potrebné používateľa neustále ale nenásilne informovať o jej stave. Vhod príde aj možnosť rýchlo vyvolať grafické rozhranie. Obidve úlohy dokáže dobre splniť ikona v hlavnom paneli (system tray).

2.7 Konzolové rozhranie

Pre uľahčenie automatizovanej správy aplikácie bude poskytnuté jednoduché konzolové rozhranie umožňujúce vykonávať základné operácie. Konzolové rozhranie nájde svoje uplatnenie hlavne pri vzdialenej správe, keď nemusí byť grafické rozhranie prístupné, alebo pri automatizovaní záloh pomocou skriptov.

2.8 Konfiguračný súbor

Prípadné hromadné nasadenie aplikácie na viacero počítačov bude uľahčené, ak bude prístupný konfiguračný súbor obsahujúci hlavné nastavenia. Inštalácia a prípadná správa aplikácie sa tak môže zjednodušiť na prekopírovanie súboru na viacero počítačov. Konfiguračný súbor bude vo formáte XML.

Kapitola 3

Návrh systému

V tejto kapitole budú popísané kľúčové rozhodnutia o návrhu systému, ktoré ovplyvnia jeho implementáciu, funkčnosť a odolnosť voči chybám.

3.1 Ukladanie zálohy

Pred samotným návrhom systému je potrebné najprv si zvoliť spôsob, akým bude systém pracovať s distribuovaným depozitárom. Výber algoritmu na správu úložného miesta je kľúčový pre odolnosť systému, jeho výkon, ale do značnej miery ovplyvní aj jeho komponenty a potrebné knižnice.

3.1.1 Viacnásobné kópie

Pravdepodobne najjednoduchším spôsobom, ako zabezpečiť bezpečnosť nášho súboru, je jednoducho ho skopírovať na viacero počítačov. Viacero kópií môžeme vytvoriť tak, že budeme jeden súbor viackrát prenášať po sieti ku každému zo zálohujúcich počítačov. Problémom pri tomto prístupe je, že nemusíme mať k dispozícií dostatočne veľké súvislé úložné miesto na každom z počítačov. Jednoduchým riešením je rozdelenie súborov na viacero menších dielov, fragmentov, ktoré následne prenesieme po počítačovej sieti na čo najväčšie množstvo počítačov.

Realizácia

Spomínaným postupným kopírovaním súborov zvýšime záťaž siete niekoľkonásobne. Tento problém bude citeľný najmä pri prenose väčších súborov. Nedá sa dosiahnuť prenosový

čas nezávislý od počtu kópií? Nakoľko prenášané budú identické kópie, môžeme využiť technológiu zvanú multicast.

> **Poznámka:** *Pre ďalšie účely práce budeme fragment chápať ako časť zlohovaného súboru.*

Multicastové adresovanie počítačov nám umožní kopírovať súbory za pomoci prostriedkov siete, pričom server rozpošle dátové pakety jediný[1] raz. Obmedzením však je nutnosť použiť na riadenie prenosu protokol UDP, nakoľko protokol TCP nie je stavaný na komunikáciu medzi viac ako dvoma bodmi, a s tým spojená potreba prostriedku garantujúceho spoľahlivý prenos nad protokolom UDP.

Čelíme teda implementácii aspoň časti funkcionality TCP nad protokolom UDP, čo však bude viac ako prácne. Potrebné je implementovať nielen mechanizmus napravujúci straty niektorých paketov, ale aj riadenie rýchlosti prenosu, spätné zoradenie paketov v správnom poradí, riadenie znovupoužitia sekvenčných čísel paketov a prípadne ďalšie. Našťastie existujú knižnice, ktoré umožňujú spoľahlivý multicastový prenos dát, je však otázne, do akej miery sa blížia svojou kvalitou protokolu TCP.

Dôsledky

Ako sa bude správať záloha v prípade výpadku niektorých uzlov našej siete? Spolu so zvyšujúcim sa počtom vytvorených záloh rastie aj pravdepodobnosť, že výpadok aspoň takého množstva počítačov, ako je počet kópií jednotlivých záloh, ovplyvní obnoviteľnosť niektorej zálohy. Odolnosť systému sa dá zvýšiť jedine väčším počtom kópií, avšak o istote môžeme hovoriť až keď bude počet kópií porovnateľný s počtom počítačov v sieti.

3.1.2 Reedove-Solomonove kódy

Reed-Solomonov kód je lineárny[2] kód založený na polynómoch nad konečnými poliami. Zaujímavou vlastnosťou tohto kódu je prispôsobiteľnosť a vysoká samoopravovacia schopnosť. Hlavným prínosom Reedovho-Solomonovho kódu je jeho rádovo väčšia efektivita v porovnaní s predchádzajúcim prístupom.

Teoretické základy

Základná idea Reed-Solomonových kódov je založená na výpočtoch nad konečnými poľami. Konečné pole $GF(q)$ pozostáva z $q = p^k$ prvkov, kde p je prvočíslo a k je kladné celé číslo,

[1] V ideálnom prípade nerátajúc náklady spojené s korekciou chýb.

[2] Súčet dvoch kódových slov je platné kódové slovo.

a je uzatvorené na operácie sčítania a násobenia. Prvky konečného poľa a operácie nad ním sú určené primitívnym polynómom nad p prvkovým poľom, jeho koreň označujeme ako α a všetky prvky poľa, okrem nuly, sa dajú vyjadriť ako mocnina koreňa modulo primitívny polynóm. Bolo dokázané, že všetky konečné polia rovnakej veľkosti sú navzájom izomorfné, teda označenie $GF(q)$ nie je závislé od výberu reprezentanta.

Majme kódované slovo $w = w_0 w_1 \dots w_{n-1}$. K tomuto slovu zostrojíme polynóm $P(x) = w_0 + w_1 x + \ldots + w_{k-1} x^{n-1}$. Kontrolný súčet k slovu w vypočítame tak, že polynóm $P(x)$ vyhodnotíme v každom prvku poľa $GF(n)$. Kontrolný súčet teda vieme použiť na zostavenie rovníc tvaru:

$$\begin{aligned} P(1) &= w_0 + w_1 + w_2 + \ldots + w_{n-1} \\ P(\alpha^1) &= w_0 + w_1\alpha + w_2\alpha^2 + \ldots + w_{n-1}\alpha^{n-1} \\ P(\alpha^2) &= w_0 + w_1\alpha^2 + w_2\alpha^4 + \ldots + w_{n-1}\alpha^{2(n-1)} \\ &\vdots \\ P(\alpha^{q-2}) &= w_0 + w_1\alpha^{q-2} + w_2\alpha^{2*(q-2)} + \ldots + w_{n-1}\alpha^{(n-1)(q-2)} \end{aligned} \tag{3.1}$$

z ktorých vieme vybrať ľubovoľných n rovníc a zostaviť tak sústavu n rovníc o n neznámych s jediným riešením.

Reed-Solomonov kód dokáže využitím spomínaných sústav rovníc opravovať až $t = \lfloor \frac{q-n+1}{2} \rfloor$ chýb. Vzniknuté chyby vieme napraviť tak, že si postupne vezmeme všetky n-tice rovníc a pokúsime sa vypočítať pôvodné slovo. Slovo, ktoré bolo v skutočnosti kódované, dostaneme ako najčastejšie opakujúci sa výsledok.

Ak však dáta chýbajú úplne, namiesto použitia chybných údajov v rovniciach (3.1) vynecháme príslušnú rovnicu úplne. Ľahko vidieť, že môžeme vynechať až $q - n$ rovníc a teda vieme korigovať výpadok až $q - n$ kódovaných znakov.

Vytvorenie zálohy

Výpočet kontrolných súčtov pre zálohu sa dá uľahčiť použitím špeciálneho tvaru Vandermodeovej matice. Nech teda

$$F = \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ \alpha^0 & \alpha^1 & \alpha^2 & \ldots & \alpha^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \alpha^{(m-1)0} & \alpha^{(m-1)1} & \alpha^{(m-1)2} & \ldots & \alpha^{(m-1)(n-1)} \end{bmatrix} \tag{3.2}$$

pričom musí platiť, že $m < q - 1$, kde q je spomínaná veľkosť použitého konečného poľa.

Ďalej nech d_1 až d_n je postupnosť prvých bajtov[3] jednotlivých fragmentov zálohovaného súboru. Obdobne c_1 až c_m je postupnosť prvých bajtov súborov obsahujúcich kontrolné súčty. Kontrolné súčty vypočítame ako

$$\begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ \alpha^0 & \alpha^1 & \alpha^2 & \dots & \alpha^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \alpha^{(m-1)0} & \alpha^{(m-1)1} & \alpha^{(m-1)2} & \dots & \alpha^{(m-1)(n-1)} \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix} \tag{3.3}$$

Nie je ťažké rozšíriť tento výpočet aj o ostatné bajty súborov, stačí len zopakovať naznačené výpočty.

> **Poznámka:** *Aby sme s fragmentmi zálohy mohli rozumne pracovať je potrebné, aby mali všetky rovnakú veľkosť. Teda je potrebné doplniť posledný najmenší fragment na veľkosť ostatných.*

Obnova zálohy

Pri obnove zálohy v prípade výpadku niektorých kontrolných súčtov opäť využijeme rovnosť (3.3).

Výpadok niektorých počítačov spôsobí nedostupnosť príslušných kontrolných súčtov. Pre každý chýbajúci súbor jednoducho vynecháme príslušný riadok z (3.3). Smelo predpokladajme, že sme stratili práve[4] $m - n$ súborov. Rovnica (3.3) sa tak zredukuje na sústavu n rovníc o n neznámych a teda k matici sústavy je možné vypočítať Gaussovou eliminačnou metódou inverznú maticu. Skutočnosť, že riešenie sústavy rovníc existuje vždy, Wicker and Bhargava (1999, chap. 1) zdôvodňujú tým, že sa pracuje so špeciálnym tvarom Vandermondovej matice, ku ktorej vždy existuje inverzná matica.

Ak $\vec{e}$ je vektor zložený z postupnosti prvých bajtov zachovaných kontrolných súčtov, tak postupnosť prvých bajtov pôvodných súborov môžeme vypočítať ako $\vec{d}^T = A^{-1}\vec{e}^T$, kde A^{-1} je spomínaná inverzná matica.

Obdobne, ako pri počítaní kontrolných súčtov, sa dá aj tento výpočet rozšíriť na celé súbory a nie len na ich prvé bajty.

Obmedzenia

Vzhľadom na to, že Reed-Solomonove kódy sú založené na výpočtoch nad konečnými poľami, kladú tieto polia aj obmedzenia pri použití týchto kódov. Počas výpočtu totiž musí

[3] Pri použití $GF(2^8)$, ak použijeme konečné pole inej veľkosti, musíme prečítať príslušný počet bitov.

[4] Pri strate menšieho počtu súborov si môžeme jednoducho vybrať, ktoré nepoužijeme.

platiť, že počet prvkov konečného poľa $GF(n)$ je väčší alebo rovný počtu kontrolných súčtov. Avšak pri použití poľa veľkosti 2^{16} môžeme pracovať s až 65 536 súbormi. Spomínaná veľkosť nám zároveň umožní pracovať s kódovými znakmi veľkosti až 16 bitov.

Oproti algoritmu viacerých kópií spomínanom v kapitole 3.1 je Reed-Solomonov kód náročnejší na výpočtový výkon. Navrhnutý algoritmus pracuje v časovej zložitosti $O(nml)$ pri vytváraní kontrolných súčtov, kde l je dĺžka jedného fragmentu, n a m sú počty fragmentov súboru a vytvorených kontrolných súčtov. Obnova zálohy pracuje v čase $O(n^2l)$. Preto nie je rozumné používať príliš veľké počty fragmentov súborov.

Výhody pri použití

Kým presne mierený výpadok množstva počítačov rovného počtu kópií záloh v sieti dokázal vyradiť prístupnosť danej zálohy, Reed-Solomonov kód s obdobným využitím miesta ostane použiteľný. Trik spočíva v tom, že pri n kópiách sme mali istý počet n-tíc, o ktoré sme nemohli prísť. Reed-Solomonov kód využívajúci rovnaké miesto môže odolať aj tisíckam výpadkov. Stačí len vhodne rozdeliť zálohovaný súbor a vypočítať dostatočný počet kontrolných súčtov.

Možnosti implementácie

Pri centralizovanom výpočte Reed-Solomonovho kódu nás pri veľkých súboroch môže limitovať časová náročnosť výpočtu. Výpočty kontrolných súčtov vieme rozložiť na viacero počítačov. Namiesto toho, aby jediný počítač vytváral všetky kontrolné súčty a následne ich distribuoval po sieti, môže tento počítač rozposlať iba jednotlivé fragmenty súborov. Každý klient schopný uchovávať súbor s kontrolným súčtom zachytí všetky fragmenty a vykoná výpočty súvisiace s korešpondujúcim výsledným riadkom v rovnici (3.3). Ak má teda klient uchovávať k-ty kontrolný súčet, tak potom vynásobí každý bajt i-teho fragmentu, ktorý zachytil, prvkom F_{ki} Vandenmordovej matice a pripočíta ho k medzivýsledku. Pri tomto postupe je žiadúce využiť postupy súvisiace so spoľahlivým prenosom spomínané v kapitole 3.1.1.

3.2 Architektúra

Pred samotnou implementáciou systému je dobré premyslieť si, ako budú vyzerať základné komponenty systému a ujasniť si prípadné nezrovnalosti. Táto kapitola poskytne popis základných komponentov systému a použitých údajových štruktúr.

3.2.1 Komunikačný protokol

Vzhľadom na to, že aplikácia je určená na prácu po sieti, je potrebné definovať komunikačný protokol pre každú činnosť zahrňujúcu súčinnosť ostatných klientov.

Sieťová komunikácia bude potrebná najmä pri vytváraní zálohy a prípadnom vyhľadávaní uložených fragmentov za účelom obnovy. Tieto typy konverzácií sú si navzájom málo podobné a je pravdepodobné, že budú musieť navzájom komunikovať rôzne komponenty vzdialených klientov a teda, že správa bude vytváraná v jednej triede a následne spracovávaná inou triedou.

Aby sa znížilo riziko inkonzistencie a chybovosti je rozumné vytvoriť knižnice obsahujúcej všetky metódy potrebné na prácu s navzájom zviazanými správami, čo nám uľahčí vytváranie odpovedí na prijaté správy ale aj údržbu aplikácie.

Iným riešením by bolo vytvorenie serializovateľných kontajnerov, avšak takýmto prístupom nedefinujeme vzťah medzi požiadavkou a odpoveďou.

3.2.2 Štruktúra zálohy

Uchovávaniu záloh v súboroch sa dá len ťažko vyhnúť. Je však potrebné definovať vzťahy medzi zálohovanými súbormi a to najmä medzi fragmentmi jednotlivých záloh a ich kontrolnými súčtami. Ďalej je spolu so zálohou potrebné uchovávať informácie o nej, aby tak bolo možné čo najviac automatizovať proces obnovy. Tieto informácie by mali minimálne obsahovať pôvodný názov súboru, čas vytvorenia zálohy, údaje identifikujúce fragmenty a kontrolné súčty patriace k zálohe a prípadne ďalšie informácie o zálohovanom súbore ako je meno vlastníka, čas poslednej zmeny či hash súboru.

Tieto informácie o zálohe je potrebné uchovávať aspoň na každom počítači, ktorý uchováva nejaké súbory zálohy, aby sa tak umožnila správa a vyhľadávanie v nich.

3.2.3 Rozloženie záťaže

Pred tým, ako uložíme jednotlivé fragmenty zálohovaného súboru a kontrolné súčty na ostatné počítače v sieti, musíme najprv určiť, ktoré počítače to budú. Ľahko vidieť, že na výber vhodných počítačov pre danú zálohu potrebujeme vedieť iba súčasný stav dostatočného množstva zúčastnených klientov. Samotný výber sa potom môže uskutočniť na základe lokálnych výpočtov bez potreby komplexného komunikačného protokolu.

Usporiadajme teda práve dostupné počítače podľa hodnoty $w(N) = \frac{r(N)*t(N)}{u(N)*b(N)+1}$, kde $w(N)$ je výsledná váha uzla, $r(N)$ je veľkosť depozitára, $u(N)$ je veľkosť už použitého

miesta a $b(N)$ je počet záloh uchovávaných na danom počítači. Funkcia $t(N)$ vracia hodnotu v závislosti od toho či je daný počítač notebook, desktop prípadne iný. Táto funkcia nám umožní odlíšiť počítače s väčšou pravdepodobnosťou výpadku. Z usporiadanej postupnosti vyberieme dostatočné množstvo počítačov s najväčšími hodnotami. Takýto heuristický prístup sa počas vývoja aplikácie javí ako dostatočný, aj keď masovejšie nasadenie výsledného programu môže pomôcť odhaliť či už lepšie parametre pre použitý vzorec, alebo značne odlišnú metódu výpočtu váhy uzla.

3.2.4 Základné komponenty

Na to, aby bola zálohovacia aplikácia plnohodnotná, musí disponovať istou množinou funkcionalít. Je rozumné rozdeliť ich do komponentov zodpovedných za ich bezproblémový chod. Medzi základné komponenty systému budú patriť:

Reed-Solomon Jadrom aplikácie bude implementácia Reed-Solomonovho kódu. Základné operácie, ktoré od implementácie očakávame sú výpočet kontrolných súčtov a obnova súboru. Výpočet Reed-Solomonovho kódu môže prebiehať dvoma základnými spôsobmi, buď lokálne alebo distribuovane. Bolo by teda žiadúce vytvoriť jednotné rozhranie schopné zastrešiť obidve implementácie.

Záloha Vytvorenie zálohy bude netriviálna činnosť zahrňujúca vyhľadanie spolupracujúcich počítačov, vybratie tých najvhodnejších z nich a riadenie komponentu zodpovedného za výpočet kontrolných súčtov. Aj keď na vytvorenie zálohy stačí zverejniť jednu-dve metódy, je to dostatočne zložitá činnosť, ktorá si zaslúži samostatnú triedu.

Obnova Svojím spôsobom opačný komponent k tomu zálohovaciemu bude zodpovedať za činnosti súvisiace s vyhľadaním, stiahnutím a obnovou zálohy.

Server Vytváranie a obnova zálohy si vyžadujú isté sieťové služby ostatných klientov. Preto je potrebný neustále bežiaci server počúvajúci na sieťovom rozhraní, ktorý bude spracovávať prichádzajúce požiadavky a tiež vytvárať a odosielať odpovede.

Lokálny depozitár Lokálne uchovávané súbory je potrebné nielen uložiť ale aj aspoň jednoduchým spôsobom prehľadávať. Taktiež je žiadúce mať prístup k štatistikám o uložených súboroch za účelom rovnomerného rozloženia záťaže. Použitie oddelenej triedy na správu depozitára nám v budúcnosti umožní aj radikálnu zmenu v prístupe k uloženým súborom bez väčších zásahov do ostatných komponentov.

Správca Aj keď sa dá funkcionalita aplikácie celkom jemne rozdeliť, jej jediným účelom je garantovanie bezpečnosti našich súborov. Pre jednoduchší vývoj rozhrania je rozumné vytvoriť adaptér, ktorý umožní jednoduchý prístup k službám aplikácie a zároveň ukryje detaily implementácie.

3.2.5 Ostatné bezpečnostné aspekty

Hlavným cieľom aplikácie je zabezpečenie integrity a dostupnosti zálohovaných súborov. Ako sa však postavíme k dôvernosti údajov?

Zabezpečenie prenosu

Ani jeden z protokolov riadiacich prenos neposkytuje možnosť šifrovať prenášané údaje. Môžeme sa síce pokúsiť o šifrovanie údajov, ale oveľa rozumnejšie je pokúsiť sa použiť protokol TLS vytvárajúci zabezpečný kanál nad dostupnými protokolmi. Otázkou však je, či bude tento protokol potrebný. Ak má byť zálohovací systém nasadený na lokálnej sieti, nie je už len samotná možnosť odpočúvať prenášané dáta príliš veľkým rizikom? Použitie VPN, kontroly prístupu k sieti a iných bezpečnostných riešení sa stáva samozrejmosťou. Preto je dôvod domnievať sa, že kým bude aplikácia používaná v súkromných sieťach, sú naše dáta v bezpečí a prípadné zabezpečenie sieťového prenosu si môžeme odložiť na neskôr.

Šifrovanie uložených súborov

Obdobne ako pri zabezpečení sieťových prenosov existujú nástroje integrované či už v operačnom systéme alebo dokonca na nižších úrovniach, ktoré pravdepodobne zvládnu túto úlohu lepšie. Pri ich použití hrozí, že šifrovaná záloha nesplní svoj základný účel, a to dostupnosť uložených údajov. Ako postupovať v prípade straty hesla? Je pre nás dôležitejšie, aby údaje nemali šancu uniknúť, aj keď niekto ukradne firemný počítač, alebo skutočnosť, že údaje môžeme kedykoľvek obnoviť? Odložme teda šifrovanie radšej stranou a zamerajme sa hlavne na dostupnosť a integritu zálohy.

Kapitola 4

Implementácia

Kapitola poskytne náhľad na implementáciu zálohovacieho systému a popíše problémy, ktorým sme čelili pri realizácií projektu.

4.1 Jazyk

Zásadným rozhodnutím pri vývoji každého projektu je výber programovacieho jazyka, ktorý značne ovplyvní naše možnosti. Vhodným jazykom pre zálohovací systém je aj jazyk Java, konkrétne verzia 1.6. Výhodami spomínaného jazyka sú platformová nezávislosť, ktorá umožní prevádzkovanie systému v rôznorodých prostrediach, ale aj fakt, že sa jedná o objektovo orientovaný jazyk umožňujúci použitie komplexných návrhových vzorov. Správne použitie návrhových vzorov nám nielen uľahčí rozhodovanie v istých situáciách a zlepší čitateľnosť kódu, rovnako umožní lepšiu modulárnosť a rozšíriteľnosť kódu. Prípadná nemožnosť použitia týchto konštrukcií pri tvorbe kódov aplikácie môže programátorovi veľmi sťažiť prácu. Ďalšou nezanedbateľnou výhodou použitia jazyka Java je dostupnosť viacerých IDE so širokou funkcionalitou urýchľujúcich vývoj.

Pri samotnom programovaní aplikácie nám príde vhod aj možnosť použiť rozhrania určujúce metódy tried bez akýchkoľvek obmedzení na ich implementáciu. Výhodou použitia rozhraní pri definovaní premenných je zlepšená udržiavateľnosť programového kódu. Pecinovský (2007) to zdôvodňuje tak, že použitie rozhrania zakrýva implementačné detaily, čím umožňuje bezbolestne nahradiť konkrétne triedy inými bez nutnosti ďalších zásahov. Pokúsme sa preto použiť a navrhnúť rozhrania tam, kde to bude možné. Na záver už len spomeňme ďalšiu dobrú programátorskú zásadu, a to snahu vytvoriť objekty zodpovedné za jedinú činnosť, čím sa nám podarí izolovať funkcionalitu do samostatných celkov a opäť

si tak zjednodušiť ďalšiu prácu.

4.1.1 Balíky

Jazyk Java umožňuje rozdeliť súbory so zdrojovými kódmi do tzv. Packages, teda balíčkov, ktoré umožňujú sprehľadniť rozloženie zodpovedností medzi zdrojové kódy. Pri rozdeľovaní tried a rozhraní do jednotlivých balíčkov máme na výber rozdeliť ich podľa vrstvy alebo podľa funkcionality.

Rozdelenia podľa vrstiev je zoskupenie tried, ktoré pracujú približne na rovnakej úrovni, teda napríklad umiestnenie komponentov zodpovedných za prístup k sieti do jedného balíčka a komponentov, ktoré pracujú nad týmito komponentmi do druhého. Uprednostníme však rozdelenie do balíčkov podľa funkcionality, keď navzájom závislé objekty zodpovedné za istú časť funkcionality aplikácie sú uložené v jednom balíčku. Táto skutočnosť umožní lepšiu modularitu aplikácie, nakoľko pridanie alebo odobratie niektorého balíčka ovplyvní ostatné balíčky menej ako pri rozdelení podľa vrstiev.

4.1.2 Externé knižnice

Na uľahčenie práce, ale aj zrýchlenie vývoja je vhodné použiť na isté úkony už vytvorené externé knižnice, ktorých je pre jazyk Java dostupných veľké množstvo. Knižnice často disponujú funkcionalitou bohatšou a lepšou ako v prípade, keď by sme si ju implementovali bez ich pomoci. Mnohé sú dokonca pod licenciami umožňujúcimi ich použitie v akademickej sfére za účelmi tejto práce.

XML

Aj keď Java poskytuje možnosti na prácu so súbormi XML a DOM, je použitie funkcii dostupných v jazyku Java v porovnaní s jazykom C# alebo php príliš komplikované. Jednoduchším riešením je použitie knižnice Dom4j voľne dostupnej na `http://dom4j.sourceforge.net/` licencovanej pod BSD License. Táto knižnica nájde svoje uplatnenie pri spracovávaní konfiguračných súborov, sieťových a iných správ.

Zaznamenávanie udalostí

Rýchly, jednoduchý, nenáročný ale aj flexibilný spôsob ako zaznamenávať vykonávané operácie poskytuje knižnica Log4j. Použitie tejto knižnice nám umožní zaznamnávať udalosti

tak, aby bol užívateľ nenásilne informovaný o stave aplikácie, ale zároveň aby mal správca prístup k podrobným záznamom o činnosti aplikácie.

Táto knižnica je dostupná na `http://logging.apache.org/log4j/1.2/` a je licencovaná pod The Apache Software License, Version 2.0.

Spoľahlivý multicast

V kapitole 3.1.1 bola spomínaná náročnosť implementácie spoľahlivého prenosu medzi viacerými uzlami za pomoci multicastového adresovania. Aj keď v skorších verziách práce boli implementované jednoduché protokoly umožňujúce takýto prenos, ukázalo sa, že pri prípadnom ďalšom vývoji je rozumnejšie použiť už existujúce prepracované riešenia. Jedným takýmto riešením je knižnica JGroups dostupná na `http://www.jgroups.org/` licencovaná pod LGPL. Táto knižnica dokonca poskytuje oveľa väčšiu funkcionalitu, ako potrebujeme pre našu aplikáciu.

4.2 Testovanie

Na základné testovanie funkčnosti aplikácie je možné použiť jednoduché techniky ako debugovanie alebo spúšťanie na testovacích vstupoch. Komplexné testovanie väčšieho množstva zložitých funkcií si však vyžiada výkonnejšiu techniku vo forme unit testov.

Vzhľadom na sieťovo orientovanú povahu aplikácie je nutné testovať jej správanie v počítačovej sieti. V takomto prípade je potrebné použiť viacero navzájom prepojených počítačov. Použitie fyzických počítačov a reálnej siete na testovanie je síce blízke reálnym podmienkam nasadenia aplikácie a poskytne tak presné výsledky. Na druhej strane však bude použitie tejto techniky veľmi prácne a náročné na zdroje.

Vhodnou náhradou väčšieho množstva počítačov je použitie ich virtuálnych náprotivkov. Virtualizačné prostredie VMware Server dostupné zadarmo na nekomerčné použitie umožňuje pokročilú správu virtuálnych počítačov a poskytuje účelné konzolové rozhranie. Práve to umožňuje automatizovať distribúciu a spúšťanie testovanej aplikácie čo opäť urýchľuje testovanie a tým pádom aj vývoj.

4.3 Realizácia

Táto časť poskytne náhľad na implementáciu samotných komponentov zálohovacieho systému a rozbor činností a postupov nimi vykonávaných.

4.3.1 Správa záloh

Lokálny depozitár

Zálohovací systém sa snaží implementovať distribuovaný depozitár. Na jeho fungovanie je však potrebné mať agentov na jednotlivých počítačov. Realizovať ich budeme za pomoci súborového systému používaného operačným systémom, a teda jednotlivé fragmenty a kontrolné súčty budeme ukladať jednoducho do samostatných súborov v zvolenom priečinku. Výhodou tohto rozhodnutia je jednoduchosť implementácie a dobrá podpora správy súborov v nami zvolenom jazyku Java. Iným riešením je použitie lokálnej databázy a ukladanie súborov do nej. Rovnako dobre sa dajú implementovať rôzne vlastné dátové štruktúry prinášajúce rôzne iné výhody.

Pre použitie obyčajných súborov pri konkurencií pokročilých dátových štruktúr hovorí nie len jednoduchosť daného prístupu, ktorá môže prísť vhod v prípade kolapsu aplikácie. V takomto prípade je veľmi uľahčená ručná obnova súborov, nakoľko prostriedky operačného systému umožnia prístup k fragmentom a kontrolným súčtom. V prípade použitia databázy by mohol byť manuálny prístup k samotným súborom zálohy veľmi obtiažny. Za svoje hovorí aj fakt, že by sme pravdepodobne museli implementovať vlastnú štruktúru, ktorá by nám poskytla žiadané výhody tým správnym spôsobom.

Údaje o súbore

Pre vyhľadanie správnej zálohy, správne fungovanie rotačnej schémy, úspešné obnovenie zálohovaného súboru ale aj mnoho ďalších činností je potrebné poznať určité atribúty pôvodného súboru. Medzi tieto atribúty patrí:

- pôvodné meno súboru
- názov zálohy
- pôvodná veľkosť súboru
- užívateľ, ktorý vytvoril zálohu
- digitálny odtlačok súboru
- čas a dátum poslednej zmeny súboru
- čas a dátum vytvorenia zálohy

- počet fragmentov súboru
- počet súborov obsahujúcich kontrolné súčty

Všetky tieto atribúty je potrebné uchovávať dlhodobo spolu so samotnou zálohou a ich strata môže spôsobiť nemožnosť obnovy zálohy. Tieto údaje majú v porovnaní so samotnou zálohou veľmi malú veľkosť, preto môžu byť uchovávané na veľkom množstve počítačov bez prílišného využitia zdrojov. Na druhej strane nie je potrebné uchovávať tieto údaje, ak nemáme prístup k žiadnym fragmentom ani kontrolným súčtom. Rozumným kompromisom je uchovávanie týchto údajov na každom počítači, ktorý uchováva aspoň jeden súbor zálohy.

Súbor s takýmito metainformáciami môžeme uchovávať vo formáte XML kvôli jednoduchosti jeho ďalšieho spracovávania. Na ukladanie nám dobre poslúži lokálny depozitár. Pomocou tohto súboru budeme schopní jednoznačne identifikovať zálohu.

Identifikácia súborov zálohy

Zatiaľ bolo spomenuté, ako sa fragmenty a kontrolné súčty budú uchovávať a tiež, čo všetko potrebujeme vedieť o príslušnej zálohe a príslušnom zálohovanom súbore. Ako však zistíme, ku ktorej zálohe patrí daný fragment a opačne, ktoré fragmenty patria k danej zálohe. Na identifikáciu zálohy je použitý súbor s metadátami, ktorý by mohol uchovávať aj zoznam príslušných súborov s ľubovoľnými názvami. Mapovanie opačným smerom by však bolo značne komplikovanejšie.

Priraďme radšej každej zálohe jedinečné meno. To môže pozostávať z názvu súboru a presného času vytvorenia zálohy. Pravdepodobnosť vzniku dvoch rovnakých názvov môžeme ešte minimalizovať pridaním IP adresy počítača, mena používateľa alebo náhodného reťazca. Každý súbor súvisiaci s touto zálohou bude mať ako prefix uvedený názov zálohy. Súbor s metadátami budeme rozlišovať na základe prípony „.xml“. Ostatné súbory, teda fragmenty a kontrolné súčty budú ako príponu používať reťazec umožňujúci jednoznačné určenie ich poradia v kódovom slove Reed-Solomonovho kódu. Ľahko vidieť, že takéto riešenie nielen jednoducho implementujeme vo zvolenom depozitári, ale umožní aj relatívne dobrú čitateľnosť človekom.

Vymazanie zálohy

Vzhľadom na povahu aplikácie môže ľahko nastať situácia, keď nebude záloha zmazaná úplne. Stane sa to v prípade, že časť počítačov, ktoré uchovávajú súbory danej zálohy,

nie je pripojená k sieti. Aj napriek tomu, že záloha môže byť v čase mazania obnoviteľná, po zmazaní prístupných fragmentov súborov ostanú tie na neprístupných počítačoch nedotknuté.

Riešením spomínaného problému môže byť plánovaná údržba repozitárov, keď sa administrátori budú snažiť maximalizovať počet dostupných počítačov počas krátkeho časového intervalu tak, aby sa každý počítač aspoň raz za čas údržby zúčastnil. Počas takejto údržby by sa zmazali všetky zálohy, ktoré nie sú obnoviteľné.

Obdobným, avšak pre používateľa menej náročným a zároveň spoľahlivejším riešením je opakované mazanie tej istej zálohy. Využijeme služby nášho časovača spomínaného v časti 4.3.4 na opakovanie lacnej operácie mazania počas intervalu dostatočne dlhého na to, aby bola záloha skutočne zmazaná.

4.3.2 Komunikačný protokol

Ako už bolo načrtnuté v kapitole 3.2.1 na výmenu riadiacich informácií budeme používať XML protokol. Popíšme teda rôzne druhy výmen správ a správy samotné viac do hĺbky.

Základné operácie riadené protokolom budú vyhľadanie vhodných počítačov na uloženie zálohy a ich informovanie o vzniku danej zálohy, vyhľadanie zálohy a zmazanie nepotrebných záloh. Na samotný prenos súborov nebudeme potrebovať XML štruktúru a ako si ukážeme v časti 4.3.3, postačíme si s oveľa jednoduchším protokolom.

Vyhľadanie kandidátov

Vyhľadanie počítačov poskytujúcich vhodné prostriedky na uloženie súborov použijeme správu obsahujúcu metadáta o zálohe zabalené do koreňového elementu `<backupRequest/>`. Odpovedať budú len tie počítače v sieti, ktoré budú schopné uchovať aspoň jeden fragment danej zálohy a v odpovedi pošlú potrebné informácie o sebe. Na základe týchto informácií budú počítače zoradené ako bolo spomínané v sekcií 3.2.3. Každý z týchto počítačov začne následne istý čas načúvať prípadným požiadavkám na súbor[1] zálohy pre prípad, že by bol zvolený za vhodného. Neúspešní kandidáti pasívne čakanie po istom čase prerušia.

Vyhľadanie zálohy podľa metadát

Obdobne, ako pri hľadaní vhodného úložiska pre našu zálohu, použijeme správu obsahujúcu metadáta zálohy zabalené tentokrát do elementu `<fragmentPoll/>`. Odpoveďou bude

[1] Treba si uvedomiť, že načúvanie na porte nie je v našom prípade drahá operácia, a teda si ju môžeme dovoliť.

správa obsahujúca nielen tieto metadáta, ale aj zoznam súborov uložených v depozitári odpovedajúceho počítača rozdelený do jedného alebo viacerých elementov `<file/>`.

Vyhľadanie zálohy podľa názvu súboru

Na zapúzdrenie tejto správy použijeme rovnaký koreňový element, ako pri vyhľadávaní podľa metadát. Rozdiel bude pochopiteľne v tom, že samotné metadáta nahradíme elementom `<filename/>`. Formát odpovede môžeme nechať rovnaký, ako v predchádzajúcom prípade.

Vymazanie zálohy

Správa prikazujúca vymazanie niektorej zálohy bude musieť opäť obsahovať metadáta zálohy. Zabalené môžu byť do elementu `<delete/>`.

4.3.3 Hlavné komponenty systému

Aj keď doteraz zmienené komponenty tvorili síce neodmysliteľnú časť systému, sú to len pasívne komponenty, ktoré pomáhajú uchovávať alebo prenášať informácie. Povedzme si teraz niečo o samotných výkonných komponentoch.

Reed-Solomon

Jadrom celého nástroja je implementácia Reed-Solomonovho kódu. V časti 3.1.2 sme si popísali teoretické aspekty fungovania spracovania tohto kódu. Popíšme si teraz bližšie, ako tento kód implementovať.

Spomínané boli dva základné prístupy k výpočtu Reed-Solomonovho kódu, a to jeho centralizovaná a distribuovaná verzia. Aj napriek tomu, že distribuovaný výpočet kódu bude pracovať v lepšom čase, je s ním spojený zásadný problém. Distribuovaný výpočet je totiž oveľa vážnejšie ovplyvnený výpadkami uzlov, ktoré sa na ňom zúčastňujú. Z tohto dôvodu sme sa rozhodli implementovať jeho centralizovanú verziu, nakoľko tak budeme schopní lepšie odolávať výpadkom dejúcim sa počas výpočtu.

Ak si chceme nechať miesto pre ďalšie zlepšovanie lokálneho depozitára, je žiadúce, aby sme vrstvu súborov pred samotnou triedou na výpočet a obnovu kódu ukryli. Vhodný nástroj na realizáciu tejto myšlienky nám Java poskytuje vo forme vstupných a výstupných prúdov a to konkrétne `InputStream`-u a `OutputStream`-u. Za pomoci potomkov týchto dvoch tried vieme čítať, respektíve zapisovať postupnosti bajtov, čo na výpočet

kódu postačuje a implementačné detaily ostanú skryté v depozitári. Prirodzeným použitím vstupných a výstupných prúdov je priradenie jedného vstupného alebo výstupného prúdu každému fragmentu a kontrolnému súčtu podľa potreby. Použitie polí alebo obdobnej štruktúry by mohlo byť v tomto prípade možno až príliš ťažkopádne a preto bude lepšie, ak použijeme návrhový vzor iterátor, ktorý nám umožní jednoduché prechádzanie vstupných a výstupných prúdov.

Pri samotnej realizácií výpočtov musíme obzvlášť dbať na veľkosť použitého konečného poľa. Aj keď môžeme implementovať konečné pole pre ľubovoľné prvočísla, najrozumnejšie je použiť polia veľkosti 2^n nakoľko tie umožnia jednoduché sčítavanie a odčítanie a taktiež sa jednoduchšie použijú pri výpočtoch na postupnostiach bitov našich súborov. Aj keď v časti 3.1.2 boli ako príklad použité prvé bajty súborov, v skutočnosti musíme používať postupnosti bitov takej dĺžky, aká je hodnota čísla n určujúceho veľkosť poľa. Táto skutočnosť je spôsobená tým, že jednotlivé prvky konečného poľa sú polynómy stupňa najviac $n - 1$ a tie sú reprezentované n bitovými postupnosťami. V prípade nesprávnej veľkosti slova by sme sa tak mohli pokúsiť o výpočet nad polynómom, ktorý do nášho poľa nepatrí, prípadne zapísať výsledok do menšieho výstupného slova, čo by spôsobilo stratu informácie. Výsledkom našich úvah o vlastnostiach Reed-Solomonovych kódov teda môže byť záver, že najvhodnejšie je konečné pole o veľkosti $q = 2^{16}$, ktorá poskytne možnosť použiť veľký počet fragmentov a bude sa jednoducho implementovať pomocou čítania a zápisu dvojíc bajtov.

Na plnohodnotnú implementáciu kódu postačujú dve verejné metódy a to metóda na výpočet kontrolných súčtov a metóda na obnovu stratených fragmentov. Obidvom na výpočet bude postačovať iterátor vstupných a výstupných prúdov a tiež počet fragmentov súboru a počet kontrolných súm.

Výpočet kontrolných súčtov bude spomínané priamočiare násobenie matíc a zápis výsledkov do výstupných prúdov pre kontrolné súčty. Obnova bude zložitejšia, nakoľko najprv je potrebné zistiť, ktoré fragmenty súborov chýbajú a vypočítať správnu maticu zodpovedajúcu matici (3.3). Na výpočet inverznej matice použijeme Gaussovu elimináciu. Po tomto kroku sme sa dostali do situácie veľmi podobnej výpočtu kontrolných súčtov, postačuje niekoľkokrát násobiť matice.

Lokálny server

Lokálne bežiaca aplikácia nie je schopná fungovať bez súčinnosti ostatných jej inštancií bežiacich v sieti. Táto spolupráca pozostáva najmä z vytvárania odpovedí na zasielané

požiadavky, prijímania a odosielania súborov. Lokálny server teda bude počúvať na TCP a UDP portoch. Pre riadiacu komunikáciu bude použitý nespoľahlivý UDP socket, nakoľko prípadné stratené pakety nepredstavujú nijakú hrozbu, v prípade príliš veľkého výpadku stačí komunikáciu zopakovať. Na prenos súborov však už je potrebný spoľahlivý protokol TCP.

Správy prijaté na UDP porte budeme jednoducho spracovávať tak, že ich najprv identifikujeme, zistíme tak, ku ktorej akcii daná správa patrí. Následne si zo správy vytiahneme údaje, ktoré použijeme na vykonanie požadovanej akcie a získanie údajov potrebných na odoslanie odpovede. Odpoveď bude zasielaná unicastom počítaču, ktorý bol zdrojom požiadavky.

Na TCP porte budeme neustále načúvať požiadavkám na prenos súborov. Samotná požiadavka je veľmi jednoduchá, súbor môžeme buď prijať, alebo odoslať. O ktorý prípad sa jedná zistíme podľa prvého riadku prenosu, ktorý bude v tvare `upload: <názov súboru>` alebo `download: <názov súboru>`. Následne buď začneme prijímať sieťovú komunikáciu a ukladať daný súbor do nášho depozitára, alebo súbor s daným názvom odošleme. V prípade komplikácií, alebo nesprávne formulovanej požiadavky spojenie ukončíme dúfajúc, že pripájajúci sa klient sa posunie na ďalší server v poradí.

Aby sme predišli konfliktom počas behu tohto servera je vhodné ho implementovať ako návrhový vzor jedináčik, ktorý zabráni spusteniu viac ako jednej inštancie triedy. Prípadný pokus o načúvanie dvoch tried na tom istom porte ako následok spustenia dvoch serverov sa skončí vyvolaním výnimky a ohrozí to stabilitu celej aplikácie. Ako rozumné riešenie sa zároveň ukazuje nápad implementovať server zodpovedajúci za spracovanie riadiacich správ a server spoľahlivého prenosu ako dve samostatné triedy.

Poller

Služby komunikačných protokolov budú využívať viaceré triedy, preto je vhodné umiestniť metódy na prácu s nimi do samostatnej triedy. Táto trieda bude ďalej využívať knižničné metódy na prácu so samotnými správami a zároveň ukryje implementačné detaily použitia týchto metód, čím sama poskytne jednoduché rozhranie na komunikáciu so sieťou.

Dôležitým aspektom pri návrhu triedy je použitie multicastu, ktorý umožní jedinou požiadavkou vyžiadať odpovede z celej siete, čo značne urýchli a zjednoduší sieťovú komunikáciu. Pri použití tejto technológie odpadá nie len povinnosť viackrát posielať také isté požiadavky, taktiež nie je potrebné vopred zisťovať adresy počítačov, ktorým bude požiadavka zaslaná.

Pre účely navrhnutého komunikačného protokolu je použitie protokolu UDP postačujúce, a teda použitie triedy so samopopisným názvom `MulticastSocket` pracujúcej nad protokolom UDP nepredstavuje žiadne výrazné obmedzenie. Čo nám ale môže spôsobiť malé starosti je obmedzenie kladené na veľkosť jedného UDP datagramu a teda aj veľkosť jednej celistvej správy. Protokol UDP totiž nie je schopný vykonať fragmentáciu príliš veľkých správ, a teda si musíme túto funkcionalitu implementovať. Na veľkosť prenášanej správy si môžeme v porovnaní s prenosom veľkého súboru položiť oveľa silnejšie predpoklady, nakoľko tieto správy nebudú dosahovať veľkosť väčšiu ako niekoľko kilobajtov[2]

Protokol riešiaci problém fragmentácie správ bude veľmi jednoduchý. Správy, ktoré sú príliš veľké na to, aby boli zaslané celé, rozdelíme na viacero častí. Na začiatok každej časti správy pridáme číslo, na základe ktorého bude možné zoradiť tieto časti do správneho poradia na vzdialenom uzle. Ďalšie spracovanie požiadaviek si zjednodučíme, ak navyše pridáme náhodné číslo, pomocou ktorého budeme môcť rozlíšiť časti rôznych správ.

Vytvorenie zálohy

Komponent zodpovedný za vytvorenie zálohy bude vykonávať postupnosť krokov vedúcu k vytvoreniu a uloženiu zálohy. Ako vstupné parametre potrebuje poznať cestu k zálohovanému súboru a tiež počty fragmentov a kontrolných súčtov ovplyvňujúce odolnosť zálohy voči výpadkom. Tieto parametre môžeme načítať z konfigurácie alebo od používateľa v prípade špeciálnych požiadaviek.

Prvým krokom v zálohovacom procese je pripravenie súboru na zálohovanie, ak totiž chceme zálohovať priečinok, je rozumné najprv vytvoriť jeho archív, čo uľahčí ďalšie spracovanie zálohy. Samozrejmosťou je vytvorenie súboru s parametrami zálohy, našich metadát. V tomto kroku sa môžeme rozhodnúť, či budeme pokračovať v ďalšom spracovaní zálohy, ak totiž existuje záloha súboru s rovnakým hashom je pravdepodobné, že sa pokúšame zálohovať jeden súbor druhý krát.

Môžeme pokračovať výpočtom kontrolných súčtov. Ďalším krokom je komunikovanie s ostatnými klientmi za účelom nájdenia vhodných lokálnych depozitárov a následne vyvolenie tých najvhodnejších z nich. Nasleduje upload súborov a prípadné odstránenie dočasných súborov. Záloha je vytvorená.

Jediný problém ktorý môže nastať je, že niektoré počítače v procese nahrávania zálohy vypadnú, v takom prípade sa musíme presunúť na ďalšie v poradí. V tomto prípade príde

[2]Odpadá teda nutnosť opätovného použitia tých istých sekvenčných čísel v priebehu jednej výmeny správ.

vhod pasívne čakanie počítačov na zálohu, nakoľko sme doteraz neboli schopní povedať, ktoré počítače sa zúčastnia na uchovávaní zálohy.

Obnova zálohy

Komponent pre obnovu zálohy umožní obnovenie zálohy zodpovedajúcej daným metadátam. Postupnosť krokov vykonávaná touto triedou bude pozostávať z vyhľadania fragmentov a kontrolných súčtov, ich následnom stiahnutí na lokálny počítač a obnovení pôvodného súboru. Vstupnými argumentmi pre obnovovaciu funkciu bude kontajner obsahujúci metadáta a súbor, do ktorého bude záloha obnovená.

Pri sťahovaní súborov zálohy sa nám vytvára miesto pre jeho optimalizáciu. Rozumné bude, ak budeme sťahovať z počítačov, ktoré sú málo vyťažené a zároveň sú dostupné cez rýchle sieťové spojenia. Kým zistenie využitia počítača je relatívne jednoduchá úloha pre lokálny server, zistenie výkonu a vyťaženosti sieťových liniek ostáva ťažkým problémom, ktoré nám navyše počítačové siete samé o sebe veľmi neuľahčia, nakoľko skrývajú svoju vnútornú architektúru.

Riadiaci modul

Ako bolo spomínané je vhodné zastrešiť vyhľadávanie, vytváranie a obnovu zálohy pod jeden komponent a použiť tak návrhový vzor adaptér. Náš riadiaci modul bude nielen vytvárať inštancie tried pre vytvorenie a obnovu zálohy, taktiež bude spúšťať aplikačný server umožňujúci distribuovaný chod aplikácie. Aby sme predišli zrážkam pri spúšťaní serveru a vytváraní záloh je žiadúce implementovať tento modul buď ako návrhový vzor knižnica alebo jedináčik, ktoré nám tiež umožnia jednoduchý prístup k hlavným metódam aplikácie.

4.3.4 Plánované zálohovanie

Hodnotu zálohovacieho systému ale aj pohodlnosť jeho používania zvýši možnosť plánovaných záloh. Jazyk Java poskytuje služby triedy `Timer`, ktorá umožní jednorázové spustenie potomka abstraktnej triedy `TimerTask` po uplynutí zadaného časového intervalu. Pre plnohodnotné využitie pre účely plánovaných záloh je však potrebné rozšíriť funkcionalitu triedy o možnosť použitia presných časov a dátumov.

Keďže opakované vykonávanie činností pozostáva len z opakovaného vykonávania jednorázových úloh po uplynutí konkrétnych časových intervalov, nebude spomínané rozšírenie predstavovať zásadný problém. Postačujúce bude, ak po vykonaní plánovanej úlohy

a taktiež po spustení aplikácie jednoducho nastavíme plánovač tak, aby vykonal ďalšiu plánovanú úlohu v poradí.

Plánovač tak umožní užívateľovi zadať deň v mesiaci alebo týždni a tiež presnú hodinu a minútu, kedy sa má zaloha vytvoriť. Časovač následne uloží zadanú úlohu do konfiguračného súboru pre časovač, aby tak bola zabezpečená možnosť pokračovať vo vykonávaní úloh aj po reštarte aplikácie. Samozrejmosťou je, že po vykonaní zmien ale aj po spustení načasovanej úlohy sa časovač znova nastaví na vykonanie ďalšej úlohy v poradí.

4.4 Používateľské rozhranie

4.4.1 Grafické rozhranie

Kým detaily fungovania aplikácie ostávajú pred používateľom skryté, grafické rozhranie bude hlavným spôsobom komunikácie s užívateľom. Rozhranie musí byť pre tak jednoduchú činnosť ako je vytvorenie zálohy veľmi jednoduché aby umožnilo intuitívne ovládanie aplikácie. Obdobne musí byť jasné a zrejmé už na prvý pohľad, ako postupovať pri obnove zálohy.

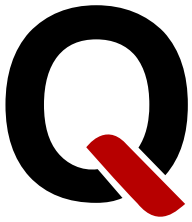
Grafické rozhranie bude preto rozdelené do dvoch sekcií a to vytvorenie a obnova zálohy, ktoré umožnia používateľovi jednoduchým spôsobom zadať potrebné parametre na vykonanie danej činnosti. V časti pre vytvorenie zálohy budú dve sekcie a to jednorázové vytvorenie zálohy a periodické vytváranie zálohy. Kým v prvej časti si používateľ vyberie súbor na zálohovanie a následne vytvorenie zálohy potvrdí, v sekcií pre periodické vytváranie zálohy si bude môcť vybrať čas alebo interval, kedy má byť záloha vytváraná.

Keďže aplikácia bude dlhodobo spustená, je dobré aby bol používateľ informovaný o jej stave pomocou ikony na hlavnom paneli (system tray) operačného systému. Po kliknutí na túto ikonu bude zobrazené okno aplikácie.

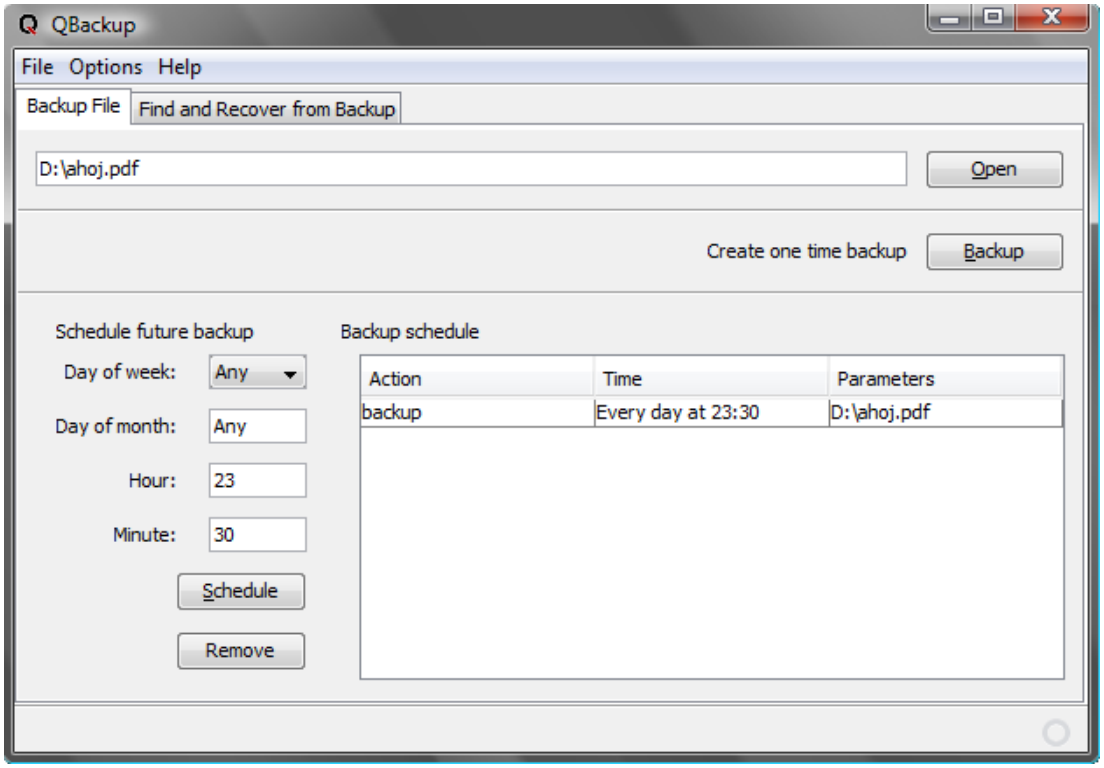
Neodmysliteľnou súčasťou aplikácie je aj jej logo. To umožní jednoduché rozoznanie aplikácie od ostatných a teda sa jej používanie stane pohodlnejším. Pre aplikáciu boli vytvorené dve verzie loga. Menšia verzia loga má slúžiť ako identifikácia aplikácie v spomínanom hlavnom paneli (system tray) ale aj ako ikona na ploche či v rôznych systémových menu. Väčšia má zase slúžiť na všeobecnú prezentáciu aplikácie.

Schopnosť dynamicky sa prispôsobovať vzniknutým novým okolnostiam v sieti je symbolizovaná červenou farbou a použitím oblého písma. Podčiarknutie loga má používateľovi evokovať pocit istoty.

Obr. 4.1: Veľká verzia loga



Obr. 4.2: Malá verzia loga



Obr. 4.3: Grafické rozhranie aplikácie

4.4.2 Konzolové rozhranie

Konzolové rozhranie nájde svoje využitie pri riadení aplikácie skriptami, čo uľahčí prácu správcom systému. Základnými činnosťami, ktoré nám toto rozhranie umožní sú: vytvorenie zálohy, obnovenie najnovšej zálohy, zmazanie najstaršej zálohy a spustenie aplikačného servera. Rozhranie taktiež umožní vyhľadať existujúce zálohy podľa názvu súboru a následne vypísať zoznam vyhovujúcich záloh.

Záver

Požiadavky kladené na implementáciu distribuovaného zálohovacieho systému sa podarilo splniť. Výsledkom je funkčná aplikácia schopná práce na viacerých počítačoch, ktorá je nielen schopná vytvoriť zálohu používateľom vybraného súboru, ale je dokonca schopná aj v prípade výpadku značného množstva uzlov túto zálohu obnoviť. Tým sa splnil základný cieľ práce implementovať systém odolný voči výpadkom.

Vďaka použitiu Reed-Solomonovych kódov je aplikácia dobre škálovateľná a tým pádom je použiteľná nielen v malých ale aj vo väčších sieťach. Vďaka použitu jazyka Java je systém platformovo nezávislý a tým pádom znižuje náklady na jeho nasadenie. Využitím multicastového adresovania je aplikácia schopná efektívne, rýchlo a jednoducho riadiť aj veľké množstvá počítačov, čo opäť zlepšuje jej škálovateľnosť. Aplikácia je tiež schopná efektívne a rovnomerne využiť prístupné úložné zdroje na zabezpečenie vysokej odolnosti voči výpadkom.

Vďaka vnútornému návrhu aplikácie ostáva voľné miesto na jej zlepšovanie a to nie len pre prípadnu lepšiu a efektívnejšiu implementáciu Reed-Solomonových kódov, ale aj rozšírenie aplikácie o odolnosť voči byzantínskym chybám. Návrh aplikácie tiež umožňuje jej využitie vo forme knižnice v iných projektoch. Rozhranie aplikácie, nie len grafické, ale aj to konzolové bolo navrhované s dôrazom na účelnosť a jednoduchosť použitia.

Práca bola pre mňa prínosná najmä preto, lebo sa mi vďaka Reed-Solomonovým kódom podarilo osvojiť si aspoň základné vedomosti z teórie kódovania a taktiež prehĺbiť si svoje vedomosti z algebry. Ďalším nezanedbateľným prínosom je skúsenosť z práce na relatívne väčšom softvérovom projekte získaná nielen pomocou viacerých rôznych a nie vždy najlepších prístupov k návrhu ale aj samotnej implementácií aplikácie. Taktiež som si rozšíril svoje vedomosti a skúsenosti s programovacím jazykom Java, a tak aj s rôznymi technikami používanými v objektovo orientovaných jazykoch ako je napríklad využitie rozhraní, vnútorných a anonymných tried. Tiež sa mi podarilo získať skúsenosť s nasadením niekoľkých základných návrhových vzorov ako je napríklad iterátor, jedináčik či adaptér. Obdobne bolo prínosné aj programovanie rôznych sieťových prvkov aplikácie.

Literatúra

Eckel, B.: 2006, *Thinking in Java*, 4rd edn, Prentice Hall, Massachusetts.

Harold, E. R.: 2004, *Java Network Programming*, 3rd edn, O'Reilly Media, Inc., Sebastopol.

Pecinovský, R.: 2007, *Návrhové vzory*, Computer Press, Brno.

Plank, J. S.: 1997, A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems, *Software – Practice & Experience* **27**(9), 995–1012.

Preston, W. C.: 2006, *Backup & Recovery*, O'Reilly Media, Inc.

Wicker, S. B. and Bhargava, V. K. (eds): 1999, *Reed-Solomon Codes and Their Applications*, John Wiley & Sons, Inc., New York, NY, USA.

Slovník pojmov

Broadcast Špeciálny prípad adresovania všetkých uzlov v sieti. Musí byť filtrovaný na hraniciach siete, inak môže spôsobiť zahltenie a nepoužiteľnosť siete.

Multicast Umožňuje adresovať viaceré uzly v sieti, ktoré sa pred tým, ako k nim budú preposielané dáta adresované týmto spôsobom, prihlásia do skupiny. Packety odosielané pomocou multicastu sú doručené výlučne uzlom z príslušnej skupiny. V sieti môže existovať viacero nezávislých skupín.

TCP Transmission Control Protocol je protokol pracujúci nad protokolom IP, ktorý zabezpečuje spoľahlivosť prenosu väčšieho množstva údajov nad týmto protokolom. Ďalej umožňuje rozlišovať jednotlivé konverzácie, riadiť rýchlosť prenosu a garantuje správne zostavenie údajov vo vzdialenom uzle.

UDP User Datagram Protocol je jednoduchý protokol, ktorý poskytuje možnosť rozlíšiť jednotlivé konverzácie, avšak neposkytuje ostatné služby protokolu TCP. Výhodou UDP sú nízke režijné nároky na sieť.

IP Internet Protocol umožňuje adresovať uzly siete a logicky ich zoskupovať. Hlavným cieľom protokolu je prenos malého množstva dát medzi dvoma uzlami.

LAN Local Area Network. Sieť lokálneho charakteru rozprestierajúca sa maximálne v niekoľkých susedných budovách. Aj napriek tomu, že často mávajú prístup k internetu, bývajú sieťe typu LAN zabezpečené pred vonkajšími zásahmi.

Zoznam príloh

Na priloženom CD sa nachádza:

- Spustiteľný súbor aplikácie
- Zdrojové kódy aplikácie
- Dokumentácia zdrojových kódov aplikácie vo formáte javadoc