Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

KIVFS - Datové úložiště

Plzeň, 2012 Radek Strejc

Originál zadání práce

- ten s červeným kulatým razítkem v jednom výtisku práce
- kopie zadání ve druhém

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 17. května 2012, Radek Strejc .....

Abstract

KIVFS - Data storage

The project KIVFS is the new experimental implementation of distributed file system. The first goal of this diploma thesis is to analyze the weak points of the data storage layer used in KIVFS on the server side from the view of requirements on the very large data storages. The second goal is to find and describe the possible ways and options for the optimalization. Following to this is another goal which is to do the implementation of the chosen options. The final aim is to test and proove that the new implemented functionalities are effective.

Obsah

1 Úvod

1.1 Projekt KIVFS

V rámci projektu KIVFS probíhá vývoj nové implementace stejnojmenného distribuovaného souborového systému (dále jen DFS) s cílem umožnit dostupnost dat na nejrozšířenějších desktopových (Microsoft Windows, GNU/LINUX) a mobilních (Windows Mobile, Android, iOS) platformách.

Hlavním důvodem vzniku projektu je neúplnost existujících DFS (jako jsou např. OpenAFS[1], Coda[2] či NFSv4[3]), která souvisí především s:

- neúplnou implementací funkcí (např. neúplná transparentnost, chybějící online replikace),
- omezenými možnostmi nasazení (např. kvůli používání NAT),

a nespokojenost s:

- malou podporou mobilních zařízení (mobilní telefony, tablety, apod.),
- složitou rozšiřitelností (např. nepřehledné zdrojové kódy),
- licenčními omezeními (např. patentová omezení, nekompatibilita GNU/GPL).

Jedním z cílů projektu je nahradit současně použitý OpenAFS na katedře informatiky a výpočetní techniky.

1.2 Cíle práce

Cílem této práce je navázat na výsledky mé dříve obhájené bakalářské práce, v rámci které bylo navrženo a implementováno základní datové úložiště KIVFS.[18] Konkrétně jde o provedení analýzy slabých míst současné verze KIVFS z hlediska požadavků na rozsáhlá datová úložiště, na kterou naváže návrh a implemenentace řešení vedoucí k jejich odstranění.

Ve výsledku práce je požadováno mít funkční a optimalizovanou implementaci datového úložiště KIVFS, která v porovnání se současnou verzí bude prokazovat jednoznačně lepší výkonnostní výsledky, stabilitu a rozšířenou funkcionalitu.

2 Obecná problematika datových úložišť

Každý uživatel či systémový správce se v souvislosti s potřebou uchovávat a spravovat různá data dříve nebo později setká s různými problémy, které se přímo týkají jeho fyzického datového úložiště. Mezi časté problémy patří:

- nedostatečná kapacita,
- bezpečnost uložených dat,
- omezená dostupnost,
- omezená výkonnost a škálovatelnost.

Problém nedostatečné kapacity datového úložiště má přímou souvislost s jeho škálováním a bezpečností uložených dat z hlediska prevence proti jejich ztrátě (zálohování) či zneužití (asymetrické šifrování). Stejně tak omezená dostupnost datového úložiště, která může být způsobená např. nemožností uživatele přistupovat ke sdílenému obsahu úložiště skrze síť vlivem nečekaného výpadku, vede k problému související s aktuálně nedostatečnou (resp. chybějící) kapacitou.

2.1 Problém nedostatečné kapacity

Nedostatečná kapacita datového úložiště vede k nemožnosti ukládání dalších dat. Může také způsobit zpomalení celého systému a v extrémním případě i jeho nepoužitelnost. Mezi časté příčiny nedostatečné kapacity lze uvést:

- stále se zvětšující objemy běžných pracovních a multimediálních dat, která je případně nutná mít zálohovaná a po čase i archivovaná (prevence proti jejich ztrátě),
- nedostupnost datového úložiště vlivem jeho nečekaného výpadku nebo v krajním případě fyzického selhání,
- omezující parametry datového úložiště, které mohou být:
  - nastavené samotným správcem (diskové kvóty, apod.) či operačním systémem (rezervace diskového prostoru, apod.),
  - dány jeho samotnou implementací (např. maximální doporučovaná velikost 2TB svazku u OpenAFS[12]).

2.1.1 Řešení v jednoduchém prostředí

V jednoduchém prostředí (např. domácnosti a malé firemní sítě), ve kterém se vyskytují jednotky uživatelů, je možné problém nedostatečné kapacity současného datového úložiště vyřešit velmi lehce. Díky stále se zvětšujícím kapacitám a zároveň snižujícím se cenám na trhu běžně dostupných datových úložišť lze stávající datové úložiště pohodlně rozšířit či zcela nahradit novým.

K realizaci škálovatelného datového úložiště je možné využít technologií:

- Logical Volume Management (LVM),
- Redundant Array of Inexpensive Disks (RAID).

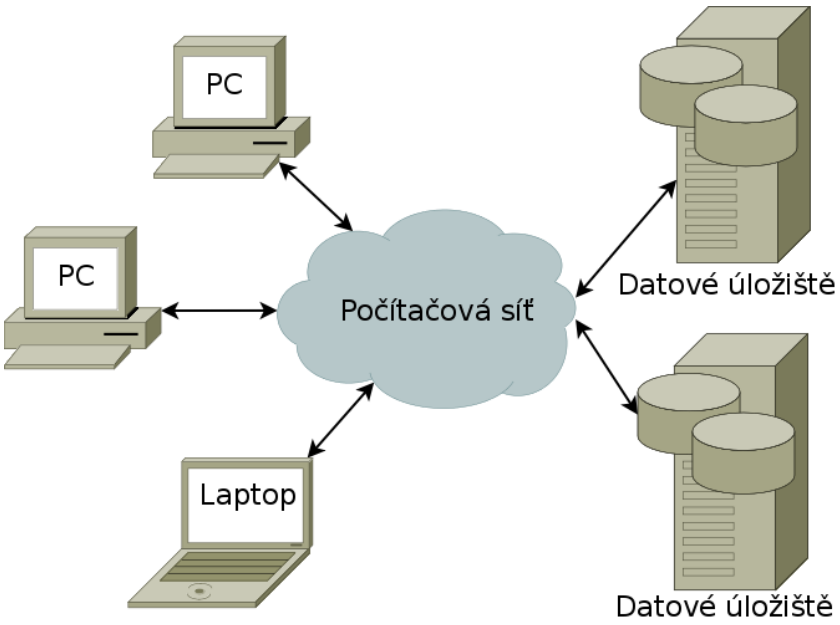
Ty umožňují vytvořit rozsáhlé datové úložiště složené z jednoho a více datových úložišť (diskových oddílů) jejich sjednocením do jednoho logického celku (oddílu) označovaného u LVM jako logický oddíl či jako pole u RAID. U obou technologií je nutná podpora operace pro změnu velikosti kapacity oddílu ze strany použitého souborového systému, které v případě velmi rozsáhlých datových úložišť může předcházet časově náročná rekonstrukce a synchronizace pole po začlenění dalšího diskového oddílu.

Dalším řešením může být využití služeb online datových úložišť. Jejich velkou výhodou je dostupnost uložených dat kdekoliv, kde je dostupné připojení k síti. Nevýhodou jsou výše nákladů za poskytovanou službu. Ty postupně přesáhnou cenu běžného datového úložiště dostupného na trhu. Mezi další nevýhody patří omezení dostupnosti uložených dat vlivem pomalé přenosové rychlosti použité linky a nutnost důvěřovat poskytovateli služby, že jí bude poskytovat spolehlivě a uložená data nebudou zneužita třetí stranou.

2.1.2 Řešení v rozsáhlém prostředí

V rozsáhlém prostředí (např. firemní a univerzitní sítě) je situace značně složitější. Z mnohem většího počtu uživatelů vyplývá i potřeba spravovat několikanásobně větší objem uložených dat, a proto rostou i nároky na celé datové úložiště týkající se jeho bezpečnosti, redundance, škálovatelnosti, apod.

Kromě lokálních úložišť jednotlivých uživatelů se proto využívají i rozsáhlá datová úložiště (obrázek č. 1), která jsou přístupná přes počítačovou síť.



Obrázek 1: Rozsáhlé datové úložiště.

3 Rozsáhlá datová úložiště

Rozsáhlá datová úložiště v síti jsou realizována několika jednotkami až stovkami fyzických datových úložišť. K jejich obsahu je zprostředkován transparentní zabezpečený přístup prostřednictvím jedné z následujících forem abstrakce sdílení dat:

- sdílení zařízení nebo jeho částí na blokové úrovni s využitím:
  - speciálních síťových komunikačních protokolů (např. iSCSI[4], DRBD[5]),
  - komunikačního rozhraní (např. Fibre Channel[6]),
- sdílení souborů skrze síťové souborové systémy (např. OpenAFS[1], Coda[2], NFSv4[3]).

Rozsáhlé datové úložiště má z důvodu vyšších zřizovacích a udržovacích nákladů (čas, finance) smysl používat v sítích větších organizací, kde jakýkoliv výpadek datového úložiště a omezení dostupnosti uložených dat může způsobit velké problémy. Z toho důvodu je u rozsáhlého datového úložiště kladen nemalý důraz na jeho:

- dostupnost, spolehlivost a výkonnost,
- centralizovanou správu,
- jednoduchou údržbu,
- škálovatelnost,

a v souvislosti s uloženými daty i na možnost jejich:

- zálohování, archivace, migrace
- zabezpečení (např. formou šifrování, omezení přístupu),
- sdílení.

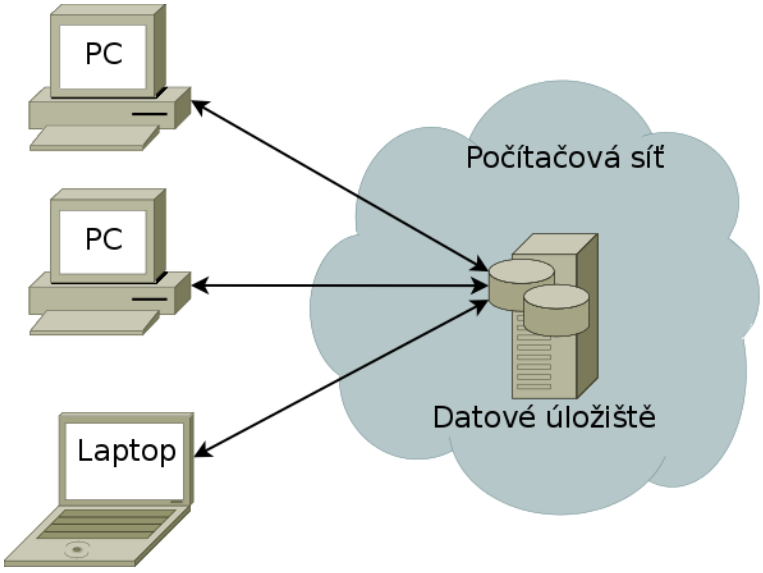
3.1 Možnosti realizace

Kvůli zmíněnému transparentnímu přístupu není z hlediska koncových uživatelů znatelný rozdíl mezi různými typy rozsáhlých datových úložišť. Z pohledu jejich správce je ale lze rozlišit dle možností realizace.

3.1.1 Centralizované datové úložiště

Přístup k celému datovému úložišti je poskytován pouze jedním přístupovým bodem v síti. Ten je dále pro zjednodušení označován jako uzel a je uvažováno, že datové úložiště, ke kterému zprostředkovává přístup, je jeho přímou součástí. Příklad je znázorněn na obrázku č. 2.

Jednoduchost centralizovaného datového úložiště, která vyplývá ze správy pouze jednoho uzlu, přináší následující výhody:



Obrázek 2: Centralizované řešení rozsáhlého úložiště.

- rychlost nasazení a okamžitá funkčnost,
- jednoduchá údržba a snadná správa.

Použití pouze jednoho uzlu je ale zároveň značně omezující. Lze hovořit o tzv. úzkém hrdle, které přináší následující nevýhody:

- omezené škálování uzlu (pouze do výšky),
- vysoké náklady na pořízení výkonného uzlu související s potřebou, co možná nejvíce oddálit časem nutnou inovaci,

a hlavně rizika:

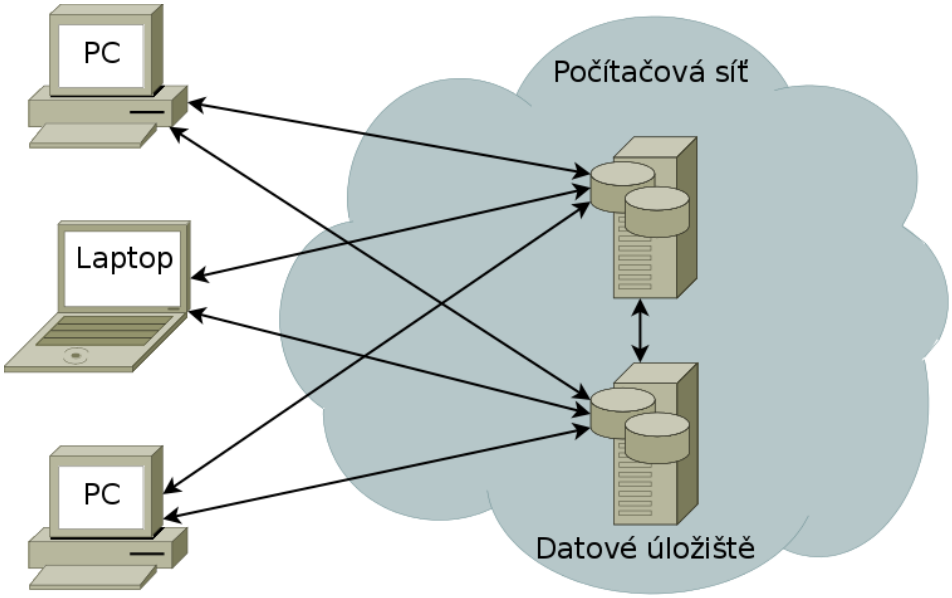
- nedostupnost všech uložených dat v případě výpadku uzlu,
- úspěšným (fyzickým) útokem získá útočník přístup ke všem uloženým datům.

Výše uvedené vlastnosti centralizovaného řešení jej předurčují k využití především v menších sítích, ve kterých možnost dočasného výpadku nepředstavuje závažné problémy.

3.1.2 Distribuované datové úložiště

V distribuovaném prostředí může být uzlů zprostředkovávajících přístup k datovému úložišti více. Každý z nich poskytuje kromě přístupu k vlastnímu datovému úložišti i přístup k datovým úložištím ostatních uzlů v síti. Z hlediska uživatele je díky tomu přístup k datům stále transparentní jako v případě centralizovaného řešení. Příklad distribuovaného datového úložiště je znázorněn na obrázku č. 3.

Srovnáním distribuovaného řešení datového úložiště s jeho centralizovanou variantou lze usoudit, že má následující výhody:



Obrázek 3: Distribuované řešení rozsáhlého úložiště.

- větší spolehlivost díky větší odolnosti vůči výpadku (po výpadku jednoho uzlu jsou nedostupná pouze ta data, která byla uložená jen na jeho datovém úložišti),
- větší dostupnost dat díky možnosti vytvořit geograficky rozsáhlé datové úložiště,
- možnost vyvažovat zátěž mezi více uzly a snižovat nároky na jejich hardware,
- téměř neomezené škálování do šířky (změna počtu uzlů neovlivní celek),
- nižší prvotní náklady na zřízení (možnost postupného rozšiřování v závislosti na aktuální potřebě).

Mezi nevýhody, které ale mohou být z určitého hlediska považovány za irelevantní, lze uvést:

- vyšší zatížení systému (synchronizační logika),
- složitost nasazení.

Distribuované řešení datového úložiště najde nejčastěji své využití v rozsáhlých sítích, které jsou rozprostřeny přes více lokalit (i geograficky velmi vzdálených), ať už z důvodů dostupnosti dat, výkonu či kapacity celého systému.

Při porovnání vlastností centralizovaného a distribuovaného datového úložiště je evidentní, že distribuované řešení odstraňuje i přes svou vyšší složitost veškeré nedostatky centralizovaného řešení.

3.2 Realizace rozsáhlého datového úložiště v distribuovaném prostředí

Rozsáhlé datové úložiště v distribuovaném prostředí lze realizovat prostřednictvím zmíněných technologií LVM a RAID, které umožní vytvořit škálovatelný diskový prostor na straně jednotlivých serverů, jež mohou využívat sdílení dat na blokové úrovni, a v kombinaci s vhodným distribuovaným souborovým systémem umožní uživatelům přistupovat k jeho obsahu na úrovni souborů.

Kombinace LVM, RAID a distribuovaného souborového systému umožní využívat všechny výše popsané výhody, které distribuované prostředí nabízí. Před samotnou realizací distribuovaného datového úložiště je nutné kromě správného návrhu jeho architektury zvolit i vhodný distribuovaný souborový systém s ohledem na jeho vlastnosti.

4 Distribuované souborové systémy

Distribuovaný souborový systém (dále jen DFS) je síťový souborový systém a zároveň případ distribuovaného systému. Ten lze definovat jako množinu nezávislých systémů (např. počítačů), které z pohledu uživatelů představují jednotný spojený systém.[7]

Stejně jako v případě běžných soubových systémů je i DFS software, který uživateli postkytuje funkce pro možnost:

- ukládat informace na datová úložiště (např. pevné disky, disková pole, apod.),
- prezentovat uložená data v podobě souborů (abstrakce uložených informací),
- organizovat uložená data do hiearchické adresářové struktury,
- manipulovat s uloženými daty srkze poskytovanou sadu operací.

Na rozdíl od běžných síťových souborových systémů poskytuje DFS výše uvedené funkce *transparentně* v distribuovaném prostředí. V dalších podsekcích jsou popsány obecné vlastnosti DFS z hlediska datových úložišť.

4.1 Transparentnost

Z pohledu uživatele je transparentnost distribuovaného prostředí, které obsahuje jedno či více nezávislých datových úložišť (serverů), nejdůležitější vlastnost DFS.

Transparentnost lze dle funkcí DFS, které jsou uživatelům právě díky ní skryty, rozdělit do následujících tříd:[7]

Transparentnost umístění. Uživatel má z libovolného přístupového bodu dostupná data ze všech datových úložišť. Nepotřebuje mít znalost, na kterém konkrétním datovém úložišti jsou uložená data, s nimiž právě pracuje.

Replikační transparentnost. Bez vědomí uživatele mohou být uložena ve více kopiích (tzv. replikách) a na více datových úložištích.

Migrační transparentnost. Data uživatele jsou stále dostupná i když se fyzicky změní umístění datového úložiště (ať už vinou výpadku či zásahem administrátora).

Přístupová transparentnost. Přístup uživatele k datům je stejný jako přístup k datům uloženým na lokálním datovém úložišti.

Souběžná transparentnost. Uživatel nemusí řešit problém souběžného přístupu více uživatelů k jednomu souboru.

Chybová transparentnost. V případě výskytu chyby (např. výpadku datového úložiště) je uživatel stále schopen pracovat se svými daty (např. díky replikaci).

4.2 Škálovatelnost

Škálování (do šířky) v DFS souvisí nejčastěji s dynamickou změnou počtu datových úložišť, která nijak neovlivní uživatele díky již zmíněné transparentnosti distribuovaného prostředí. V rámci celého systému má škálování vliv na jeho výkonnost a celkovou kapacitu.

4.3 Jmenný prostor

Identifikace objektů (souborů) a příslušných zdrojů (obsahu souborů) je v DFS zajištěna pomocí mapování názvů souborů (organizovaných v adresářové struktuře), se kterými klient pracuje, na adresy fyzicky uložených datových bloků či přímo souborů na datových úložištích. Překlad názvů je realizován na straně serveru, který spravuje tzv. jmenný prostor.

4.3.1 Hiearchický jmenný prostor

V DFS jsou soubory v rámci jmenného prostoru organizovány v hiearchické (adresářové) struktuře. Transparentnost přístupu je zajištěna díky možnosti na straně klienta lokálně připojit vzdálený svazek, jehož adresářová struktura (a soubory v ní uložené) odpovídá adresářové struktuře organizované v rámci jmenného prostoru, který je spravován serverem.

Jmenný prostor lze klasifikovat dle způsobu, kterým je klientům poskytován, do následujících tříd:[7]

Soukromý jmenný prostor. Každý klient má přístup pouze k vlastnímu jmennému prostoru. Nevýhodou je téměř nemožné sdílení souborů v něm organizovaných. Řešením je pevně určit neměnnou část jmenného prostoru, ke které má přístup více uživatelů.

Globální jmenný prostor. Celý jmenný prostor je sdílen a dostupný všem klientům. Ti mají možnost omezením přístupu ostatním uživatelům určit, která část odpovídá jejich lokálnímu jmennému prostoru.

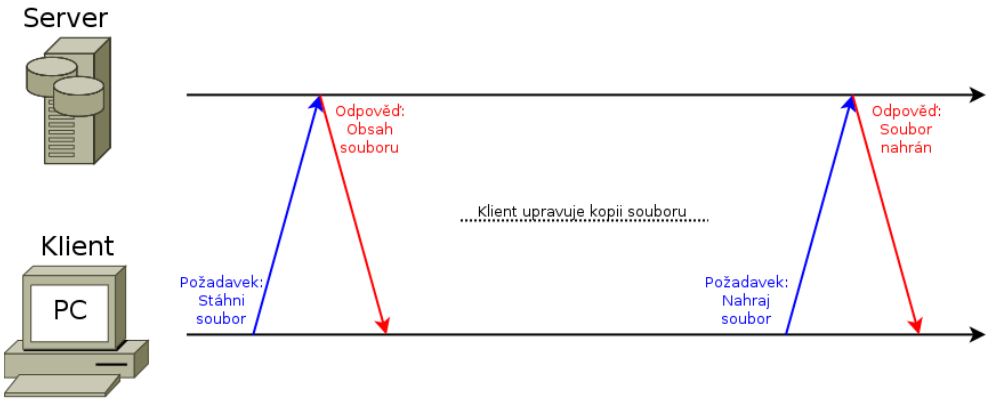
4.4 Přístup k souborům

Uživatelům DFS je umožněn přístup k uloženým datům formou abstrakce sdílení souborů. Tu lze implementovat pomocí dvou základních modelů: [10]

Upload/download model. Klient před použitím souboru stáhne celý jeho obsah ze vzdáleného datového úložiště a uloží jej do lokálního datového úložiště. Po případných úpravách souboru nahraje celý jeho obsah zpět na vzdálené datové úložiště (obrázek č. 4).

Výhodou modelu je jeho jednoduchá implementace. Za jeho nevýhody lze považovat:

- klient musí disponovat dostatečnou kapacitou lokálního úložiště (paměti),
- náročné na datové přenosy při častých změnách souboru.

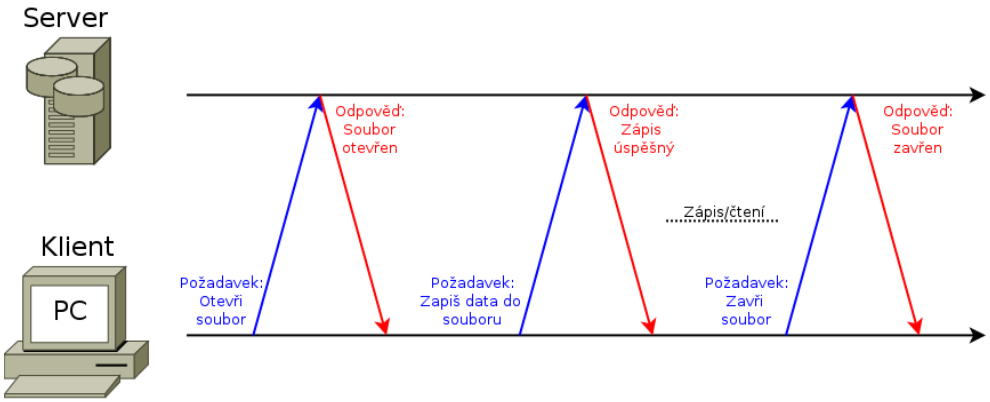


Obrázek 4: Upload/download model.

Model vzdáleného přístupu. Klient k souboru přistupuje skrze rozhraní vzdáleného datového úložiště (obrázek č. 5). Pracuje vždy jen s blokem dat (částí souboru), které je potřeba stáhnout nebo upravit. Veškeré operace se souborem jsou implementovány na straně serveru.

Model značně snižuje nároky na datové přenosy a kapacitu lokálního datového úložiště na straně klienta. Za nevýhody modelu lze považovat:

- vyšší režii na straně serveru,
- složitost implementace.



Obrázek 5: Model vzdáleného přístupu.

4.5 Stavové a bezestavové servery

Stavovost serveru lze definovat jako jeho schopnost udržovat vazbu mezi příchozími požadavky od klienta.

Stavový server uchovává informace o vzájemně souvisejících požadavcích. Jednotlivé požadavky lze mezi sebou provázat a v závislosti na celkové informaci zpracovávat. Oproti tomu bezestavový server neuchovává žádné informace o příchozích požadavcích. Každý požadavek je zpracován zcela samostatně a musí proto obsahovat veškeré informace nutné ke zpracování. Srovnání vlastností stavového a bezestavového serveru je uvedeno v tabulce č. 1.

	Stavový server	**Bezestavový server**
Výhody	Menší nároky na datové přenosy Nižší režie serveru Možnost zamykání souborů	Odolné vůči výpadku
Nevýhody	Složitá obnova po výpadku	Vyšší režie serveru Větší nároky na datové přenosy

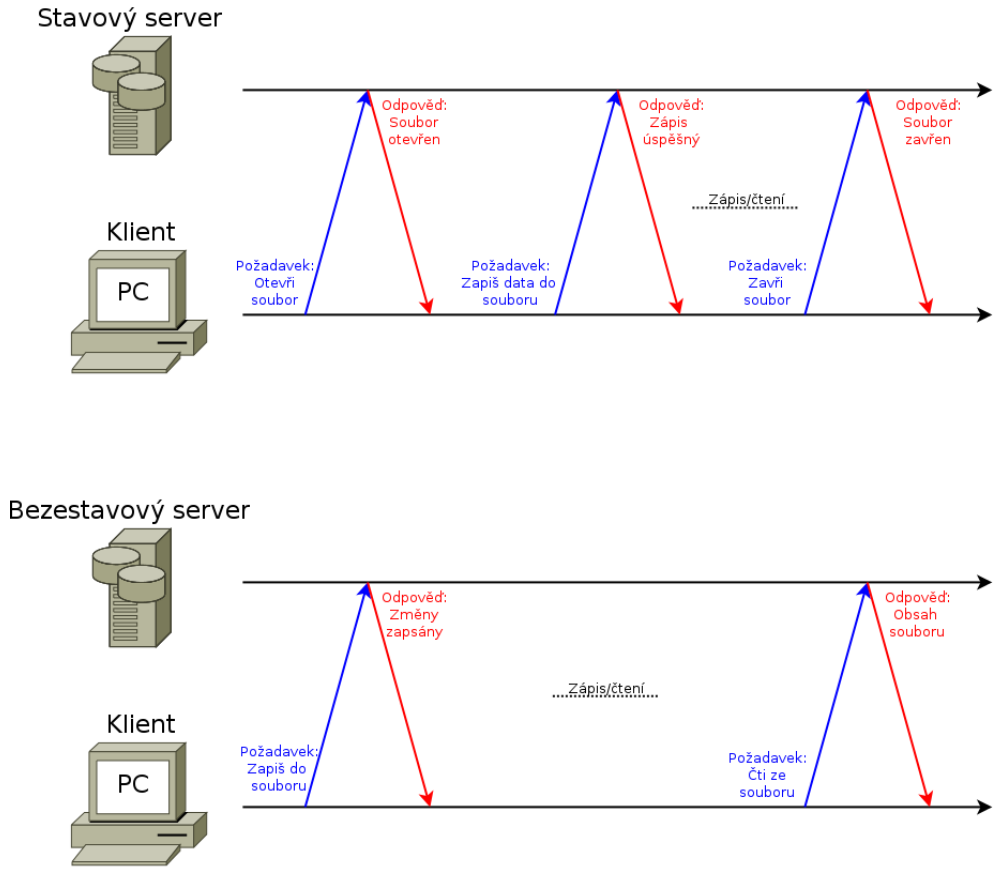
Tabulka 1: Srovnání vlastností stavových a bezestavových serverů

Jako typický příklad (obrázek č. 6) demonstrující rozdíly mezi stavovovým a bezestavovým serverem lze uvést zpracování požadavků na přístup ke vzdálenému souboru. Ten je možné pomocí stavového serveru realizovat jako postupné zpracování požadavků na:

1. otevření souboru,
2. provedení operací nad otevřeným souborem,
3. uzavření souboru.

Požadavky na přístup k souboru zpracovávané bezestavovým serverem obsahují veškeré informace nutné k jeho jednorázovému zpracování a provedení požadované operace:

- identifikátor souboru,
- parametry operace.



Obrázek 6: Přístup k souboru umístěném na stavovém a bezestavovém serveru.

4.6 Replikace dat

Data uživatelů mohou být v rámci DFS uložena ve více kopiích (tzv. replikách) a na více datových úložištích. Replikace přímo souvisí se škálováním systému a jejím cílem je zvýšit jeho:[8]

Výkonnost. Data mohou být poskytována z více zdrojů, které spravují jejich repliku, a v rámci celého systému může být vyvažována zátěž.

Dostupnost. Data mohou být poskytována z nejvhodnějšího (nejbližšího) zdroje spravujícího jejich repliku.

Spolehlivost. Při výpadku mohou být data stále dostupná z dalšího zdroje spravujícího jejich repliku.

4.6.1 Konzistenční modely

Správa více replik vede k problému v zachování jejich konzistentního stavu. Úprava jedné z replik vede k nekonzistentnímu stavu ostatních. V distribuovaných systémech

je tento problém řešen prostřednictvím použití modelů konzistence. Ty definují pravidla, jejichž dodržení zajistí zachování konzistentního stavu v rámci celého distribuovaného datového úložiště.[7]

V rámci DFS se používají datacentrické konzistenční modely, jejichž cílem je zajistit konzistentní stav replik i po vykonání souběžných operací pro zápis a čtení. Datacentrické modely lze rozdělit do dvou skupin podle toho, zda používají synchronizační operace. Ty slouží k synchronizaci všech lokálních kopií dat (replik) propagováním lokálních změn nebo přijímáním změn z ostatních kopií.

Modely nepoužívající synchronizační operace:

Striktní konzistence. Striktní uspořádání operací. Při každém čtení dat ze souboru je vrácena hodnota odpovídající poslednímu zápisu do něj. Operace na zápis jsou okamžitě a atomicky vykonány ve všech kopiích. To značně snižuje výkonnost celého systému. Jediným možným řešením je snížit požadavky na konzistenci (tzn. použít slabší model konzistence).

Lineární konzistence. Sdílené přístupy, které jsou uspořádány dle globálních (neunikátních) časových značek, jsou viděny všemi procesy ve stejném pořadí.

Sekvenční konzistence. Sdílené přístupy, které nejsou uspořádány v čase, jsou viděny všemi procesy ve stejném pořadí.

Příčinná konzistence. Všechny procesy vidí příčinně související sdílené přístupy ve stejném pořadí.

FIFO konzistence. Všechny procesy vidí zápisy ostatních procesů v pořadí, ve kterém byly provedeny. Zápisy od různých procesů nemusí být vždy viděny ve stejném pořadí.

Modely používající synchronizační operace:

Slabá konzistence. Sdílená data mohou být považována za konzistentní až po dokončení synchronizace.

Uvolňující konzistence. Sdílená data jsou označena za konzistentní po opuštění kritické sekce.

Vstupní konzistence. Sdílená data náležející kritické oblasti jsou označena za konzistentní při vstupu do kritické sekce.

4.6.2 Konzistenční protokoly

Implementace jednotlivých modelů popisují konzistenční protokoly. Ty definují, jak je prováděna samotná distribuce změn na všechny repliky, kterou realizují tzv. replikační manažeři.

Dle způsobu, kterým replikační manažeři (dále jen RM) zajišťují konzistentní stav celého distribuovaného datového úložiště, lze replikaci rozdělit do dvou typů:[8]

Master/slaves replikace. V daném čase je dostupný pouze jeden primární RM (master) a jeden nebo více vedlejších RM (slaves). Operace na zápis zpracovává

master RM a posílá kopie změněných dat všem slave RM. Operace v případě, že master RM vypadne, jeden ze slave RM je zvolen jako nový master.

Multimaster replikace. Všichni RM jsou si rovni. Kterýkoliv z nich může zpracovávat operaci na zápis a následně distribuovat změny, které jsou nezávisle (ale identicky) zpracovány ostatními RM. V případě výpadku libovolného RM nedojde k ovlivnění ostatních.

4.7 Klienské cache a offline operace

Speciální vlastností DFS je podpora offline operací. Nemá-li uživatel přístup k síti, může se svými daty stále pracovat díky jejich kopii v lokální cache klienta. V závislosti na její kapacitě je uživateli zajištěna stálá dostupnost jeho pracovních dat. Lokální cache na straně klienta je také využívána ke zvýšení výkonnosti práce se soubory, které jsou v ní uložené.

Kvůli zajištění konzistentního stavu souborů v rámci celého DFS dojde po dokončení jejich úprav (zvláště pak s použitím offline operací) v klientské lokální cache k navázání spojení mezi klientem se serverem, po kterém následuje synchronizace změněných souborů.

4.7.1 Správa verze souboru

Aby klienti byli schopní zjistit, který soubor byl právě vytvořen či upraven, musí obdržet informaci, že došlo ke změně stavu souboru s daným názvem. Průběžné dotazování se serverů nebo zasílání informovácí klientům o změně obsahu souboru s daným názvem je ale značně neefektivní. Tento problém řeší v DFS správa verzí souborů.[10]

Správa verzí souborů, resp. jejich replik, je velmi jednoduše realizovatelná implementací čítačů změn jednotlivých souborů. Replika souboru s nejvyšší hodnotou čítače změn má nejvyšší verzi a lze být považována za aktuální. Ostatní repliky souboru jsou označeny jako nekonzistentní a jejich aktualizace je automaticky provedena v závislosti na implementovaném modelu konzistence.

Požadavek na otevření souboru je vždy proveden nad aktuální verzí repliky souboru. V případě, že obsah repliky souboru byl změněn, při zpracování požadavku na zavření repliky souboru je zvýšena její aktuální verze. V případě, že je implementován model přístupu k souboru typu upload/download a nebo změny souboru proběhly pouze v lokální cache klienta, může dojít při vykonávání operace zavření repliky souboru ke konfliktu s verzí repliky souboru uložené na straně serveru, jejíž obsah byl v nedávné době změněn jiným klientem. Mezi základní řešení možných konflitů lze z pohledu klienta uvést:

Ignorování konfliktu. Obsah souboru (resp. jeho repliky) na serveru je přepsán nehledě na jeho verzi.

- riziko ztráty provedených změn v souboru na straně serveru

Sjednocení změn. Obsah souborů je porovnán a následně jsou vyhodnoceny jejich rozdílné části, které odpovídají provedeným změnám. Řešením je sjednocení

těchto změn a verzí, po kterém následuje další pokus o uložení souboru na server. Změny stejných částí souboru (kolize) jsou vyřešeny jejich serializací.

- kolize musí vyřešit uživatel, který rozhodne, jaké změny se mají zachovat

Zásah uživatele. Uživatel rozhodne, která verze souboru se má považovat za aktuální. Rozhodne-li, že aktuální verze souboru má být jeho lokálně upravená kopie, aktualizuje se její verze a následuje další pokus o další uložení souboru na server.

- uživatel musí být na konflikt upozorněn a vyřešit jej

4.8 Sdílení souborů

Sdílení souborů umožňuje přistupovat více klientům k jednomu souboru. Pro uživatele je ideální stav, kdy jsou veškeré provedné změny okamžitě viditelné pro ostatní. V DFS je tento stav nedosažitelný z důvodů možné latence sítě, která způsobuje zpoždění zápisů všech změn během synchronizace replik. Souběžný přístup v DFS k jednomu souboru a provádění konfliktních operací (pro zápis) může proto vést k problémům týkajících se konzistentního stavu souboru.

Řešení problému v rámci DFS je závislé na implementované sémantice sdílení souborů a metodě vyloučení souběžného přístupu k souborům.

4.8.1 Sémantika sdílení

Sémantika sdílení souborů definuje, kdy jsou změny provedné jedním uživatelem viditelné i pro ostatní uživatele. Mezi nejčastěji implementované patří:[10]

Unixová sémantika. Cílem je zajisit aktuálnost souboru. Každá operace pro zápis do souboru je atomicky a okamžitě provedena ve všech jeho kopiích. Operace čtení ze souboru vrátí vždy hodnotu poslední zápisu do něj. Klient, který provedl operaci pro zápis do souboru musí před provedením další operace čekat než je zápis proveden ve všech kopiích souboru.

Transakční sémantika. Cílem je zajisit konzistentní stav souboru. Výsledky operací pro zápis jsou v během transakce průběžně ukládány (v pracovním úložišti) a permanentně (atomicky) potvrzeny při splnění podmínek konzistence na konci transakce.

Relační (session) sémantika. Cílem je zajistit efektivní přístup k souboru. Výsledky operací pro zápis jsou ukládány do pracovní kopie a permanentně potvrzeny při ukončení relace.

Jednoduchým způsobem, kterým lze také vyřešit problémy souběžného přístupu k souborům, jsou nezměnitelné soubory. Soubor nelze otevřít pro zápis, ale jen vytvořit a následně číst jeho obsah. Sdílení souboru a i jeho replikace je díky tomu značně zjednodušená. Nevýhodou jsou však vyšší nároky na celkovou kapacitu datového úložiště.

4.8.2 Vyloučení souběžného přístupu pomocí zamykání

Je-li server implementován jako stavový, může u souborů uchovávat stav jejich zámků. Ty slouží vyloučení souběžného přístupu k souboru, během kterého by došlo ke konfliktním změnám v jeho obsahu. V případě, že je soubor uzamčen, pak nemůže být otevřen pro zápis.[8]

Zamykání souborů v DFS lze realizovat implementací čítačů, jejichž nenulová hodnota odpovídá zamknutí souboru. Mezi základní druhy zámků patří zámky pro čtení a zápis. Platí, že při:

- vícenásobném čtení nedochází ke souběhu konfliktních operací,
- vícenásobném zápisu může dojít k souběhu konfliktních operací,
- vícenásobném čtení a zápisu může dojít k souběhu konfliktních operací.

V tabulce č. 2 je popsána kompatibilita zámků, která určuje, kdy je možné soubor zamknout pro čtení či zápis.

Stav zámků souboru	Lze zamknout pro čtení?	Lze zamknout pro zápis?
Odemčený	Ano	Ano
Zamčený pro čtení	Ano	Ne
Zamčený pro zápis	Ne	Ne

Tabulka 2: Zamykání souboru (kompatibilita zámků)

4.8.3 Vyloučení souběžného přístupu pomocí předávání pověření

Pověření povoluje klientovi, který jej obdžel, přistupovat k obsahu souboru. Ostatní klienti k obsahu souboru přistupovat nemohou. Klient po dokončení úprav předá pověření dalšímu čekajícímu zájemci o přístup k souboru a to buď formou pověřením následujícího klienta (čekatele) v logickém kruhu, který je složen ze všech klientů, nebo předáním prvnímu klientovi z fronty čekatelů, do které se klienti v případě zájmu o přístup k souboru zařazují.

4.9 Bezpečnost uložených dat

V DFS je nutné zajist bezpečnost uložených dat prostřednictvím omezení přístupu k nim. Toho lze docílit pomocí:[8]

Ověření uživatele. Po navázání spojení mezi klientem a serverem dojde k ověření identity uživatele. V případě úspěšného ověření je uživateli povolen přístup k obsahu datového úložiště. V opačném případě je mu přístup zamítnut.

Šifrování komunikace. Komunikace mezi klientem a serverem je šifrována dle předem dohodnutého šifrovacího algoritmu. Nelze tak jednoduše odposlechnout posílaná data, příkazy ani přístupové údaje.

Seznamu přístupových práv. Ke každému souboru lze přiřadit seznam přístupových práv (dále jen ACL) pro jednotlivé uživatele nebo skupiny uživatelů. Přístupová práva definují operace, které je možné nad souborem vykonávat. Lze tak např. nastavit, že uživatel je vlastník souboru a může se souborem libovolně manipulovat (má práva pro čtení i zápis). Ostatní uživatelé např. mohou jen číst obsah souboru (mají práva pouze pro čtení).

Šifrování obsahu souborů. Šifrováním obsahu uložených dat na datovém úložišti lze účinně chránit uložená data uživatelů před fyzickým útokem na ně (např. krádež serveru). I v případě jeho úspěšné realizace jsou pak utočníkovi získaná data zcela bezcenná, neboť není schopen je dešifrovat a následně přečíst.

5 Distribuovaný souborový systém KIVFS

KIVFS je experimentální implementace distribuovaného souborového systému. Hlavní cíle projektu jsou:[13]

- úplná transparentnost distribuovaného prostředí,
- funkčnost v téměř libovolném síťovém prostředí,
- multimaster replikace (dostupnost všech replik pro čtení i zápis),
- vysoká škálovatelnost, výkonnost, stabilita a bezpečnost,
- podpora klientské cache a offline operací,
- podpora mobilních platforem,
- modulárnost (snadná rozšiřitelností a modifikovatelnost),
- nezávislost na použitých technologiích.

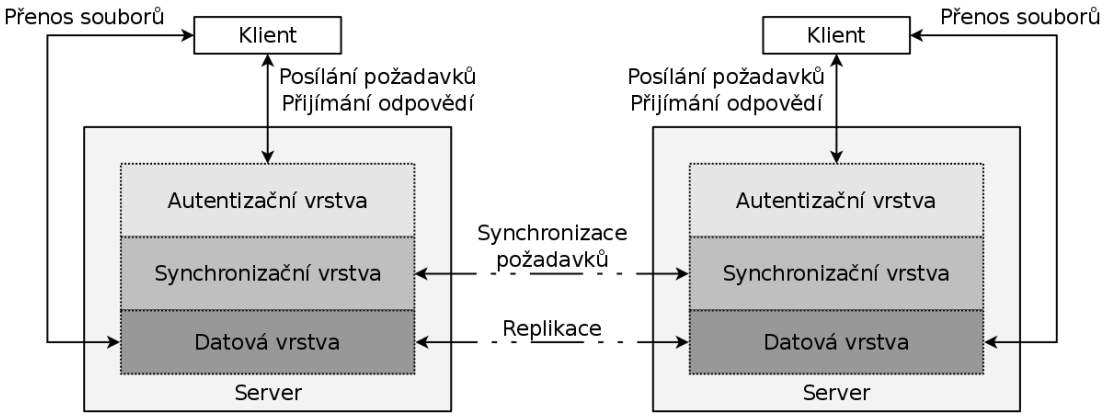
5.1 Architektura

Distribuovaný souborový systém KIVFS je založen na síťové architektuře klient/server. Veškerá komunikace probíhá prostřednictvím vlastního protokolu KIVFS, který je:[18]

- postaven na spolehlivém protokolu TCP/IP, díky kterému není nutné implementovat vlastní mechanismus pro potvrzování zpráv způsobující vyšší režii a zpoždění při zpracování požadavků,
- binární a optimalizovatelný z hlediska výsledné velikosti přenášených zpráv, na kterou jsou citlivé zvláště klientské aplikace určené pro mobilní platformy,
- založen na principu posílání paketu složeného z hlavičky obsahující informace o požadavku (resp. odpovědi) a těla obsahujícího data (obrázek č. 8).

Klient může být připojen k libovolnému serveru (obrázek č. 7), který mu zprostředkovává přístup k datům z celého systému a umožňuje jejich správu (transparentnost umístění, replikace, migrace, souběhu a chyb). Získaná data dále prezentuje v závislosti na cílové platformě (transparentnost přístupu a chyb) uživateli např. jako:

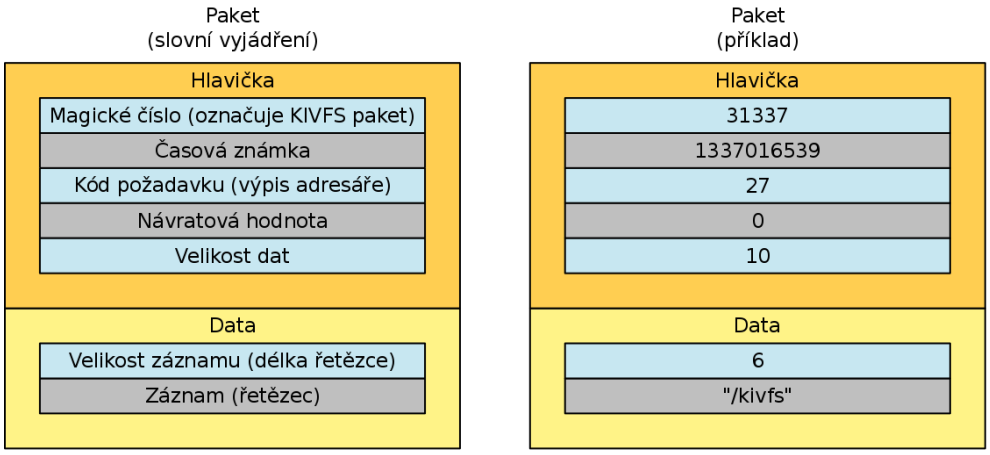
- připojený vzdálený svazek s využitím jaderného modulu KIVFS[21] pro operační systém GNU/Linux,
- připojený svazek s využitím jaderného modulu FUSE[22],
- vzdálený svazek (zásuvný modul do souborového manažeru[23]).



Obrázek 7: Architektura KIVFS.

Server je rozdělěn dle funkcionality do jednotlivých na sobě nezávislých modulů dále označovaných jako vrstvy, které spolu komunikují a spolupracují. Jednotlivé vrstvy jsou tvořeny samostatnými serverovými aplikacemi a jsou naprogramované s využitím programovacího jazyka C. Cílovou platformou pro použití serveru je operační systém GNU/Linux. Po případných úpravách, které spočívají především nahrazení použitých knihoven, je možné server použít i na jiné desktopové platformě (např. OpenBSD, Microsoft Windows).

Modulární architektura umožňuje rozšiřitelnost celého systému o další funkcionalitu a to v podobě přidávání dalších modulů. V případě potřeby je možné upravit či zcela nahradit modul bez ovlivnění funkčnosti zbytku systému.



Obrázek 8: Obsah paketu protokolu KIVFS (požadavek na výpis adresáře).

5.1.1 Autentizační vrstva

Autentizační vrstva v KIVFS slouží jako přístupový bod do celého systému pro klientské aplikace. Kvůli zabezpečení komunikace je veškerá výměna zpráv šifrována pomocí alogritmů, které poskytuje knihovna OpenSSL.

Po připojení klienta zajišťuje jeho autorizaci pomocí vlastní databáze uživatelů a systému Kerberos. Důvodem pro použití vlastní databáze je možnost omezit přístup uživatelům, kteří sice mají účet v systému Kerberos, ale nemají zřízený účet v rámci KIVFS. V případě, že klient je úspěšně autorizován, autentizační vrstva předává veškeré jeho požadavky k vyřízení synchronizační vrstvě. V opačném případě ukončuje spojení.[19]

5.1.2 Synchronizační vrstva

V rámci KIVFS může být více klientů pracujících se stejnými daty připojeno k různým přístupovým bodům. V případě, že se požadavky v rámci celého systému nepropagují dostatečně rychle (ideálně bez zpoždění) a ve stejném pořadí, mohlo by dojít k souběžnému přístupu a nekonzistentnímu stavu uložených dat. Tento problém řeší synchronizační vrstva, která požadavky na jakokouliv změnu dat s využitím Lamportových hodin a dalších algoritmů synchronizuje mezi všemi právě dostupnými servery.[20]

Po úspěšné synchronizaci požadavků je předá ke zpracování datové vrstvě. Mezi další úlohy synchronizační vrstvy patří:

- detekce výpadku serveru a zajištění jeho obnovy pomocí synchronizace jeho uložených dat s daty z ostatních serverů,
- průběžné vyhodnocování stavů linek mezi servery a nejlepší cesty k distribuci požadavků mezi servery,
- řízení transakcí a správa logických hodin v systému.

5.1.3 Datová vrstva

Datová vrstva slouží jako datové úložiště pro všechny informace, které jsou nutné pro správnou funkci celého systému, a zajišťuje jejich kompletní správu.[18] Předmětem této práce je optimalizace datového úložiště a to je proto podrobněji rozebráno v následující sekci 6.

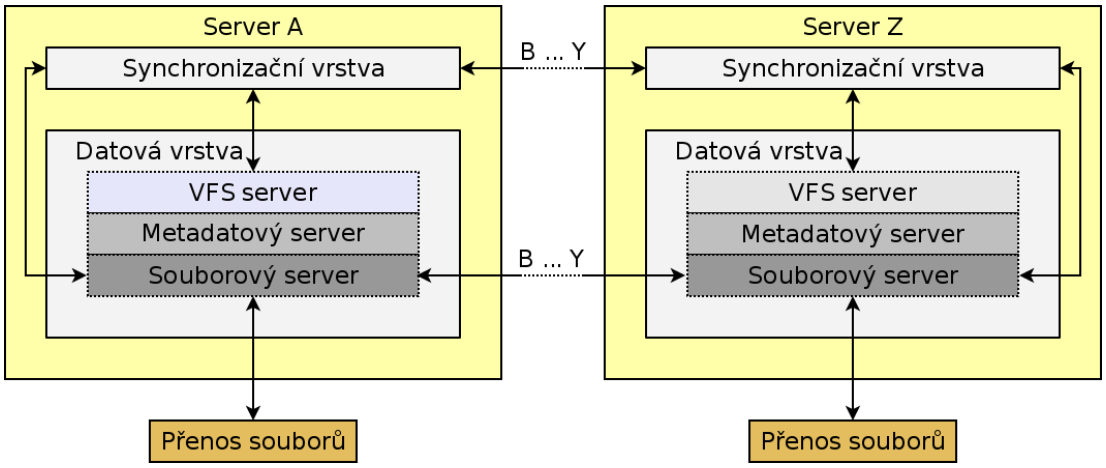
6 Datové úložiště v KIVFS

Datové úložiště (obrázek č. 9) je v rámci jednoho KIVFS serveru tvořeno třemi samostatnými serverovými aplikacemi:[18]

VFS server. Slouží jako rozhraní mezi synchronizační vrstvou a metadatovým nebo souborovým serverem. V závislosti na druhu požadavku jej předává ke zpracování právě jednomu z nich. Zajišťuje nezávislost vyšších vrstev na implementaci samotného datového úložiště.

Metadatový server. Zprostředkovává spojení s relačním databázovým systémem MySQL a spravuje veškerá v něm uložená data. Mezi ty patří informace o uložených souborech (tj. metadata), jmenném prostoru, uživatelích, ostatních serverech, atd. Díky použité databázi k uchovávání všech informací lze označit server jako stavový.

Souborový server. Spravuje fyzický obsah jednotlivých souborů uživatelů a zprostředkovává jim jejich přenos. Ve spolupráci s metadatovým serverem a všemi ostatními souborovými servery zajišťuje replikaci.



Obrázek 9: Datové úložiště v KIVFS.

Organizace a správa uložených dat pomocí jednotlivých serverů a relačního databázového systému umožňuje:

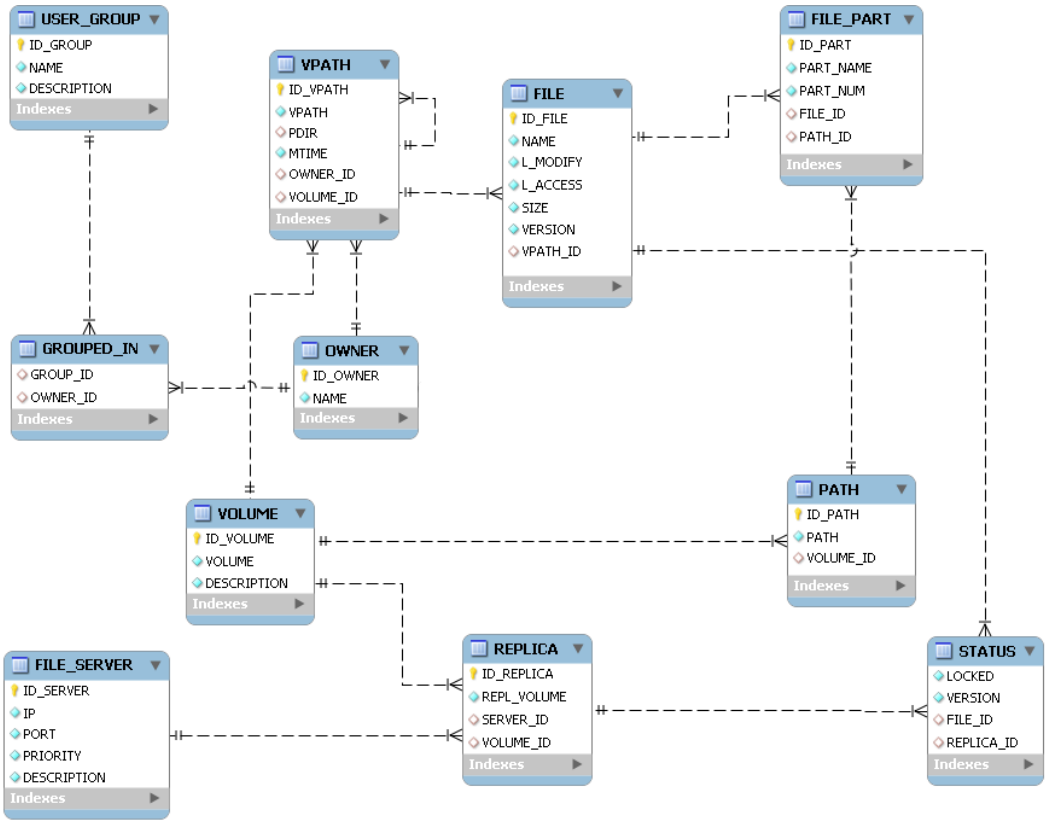
Snížit počet přístupů k fyzickému datovému úložišti. Diskové operace jsou časově mnohem náročnější než čtení informací z databáze, kterou je možné alespoň částečně uchovávat v paměti. Proto je snaha minimalizovat počet fyzických přístupů na svazky (např. připojených pevných disků) souborových serverů, ke kterým dochází během přenosu obsahu souborů nebo jejich replikaci, a dosáhnout tak co možná nejvyšší rychlosti a nízkého zatížení.

Minimální velikost databáze. Relační model dat umožňuje efektivně organizovat velké množství uložených dat, mezi které patří již zmíněná metadata,

informace o uživatelích, ACL, apod. a efektivně tak minimalizovat jejich redundanci. Vzhledem k tomu, že obsah souborů spravují souborové servery, velikost databáze nikdy nebude příliš velká i když v systému bude uloženo mnoho souborů. Díky tomu je možné, aby každý KIVFS server mohl spravovat vlastní kopii celé databáze a měl tak vždy dostupné kompletní informace o celém systému (synchronní stav všech databází zajišťuje synchronizační vrstva).

Jednotný přístup k uloženým informacím v databázi. k veškerým datům uloženým v databázi lze přistupovat pomocí dotazovacího jazyka SQL. Díky tomu je možné uložená data efektivně spravovat a vyhledávat v nich požadované informace.

Důvodem pro použití relačního databázového systému MySQL je jeho rychlost při práci s daty.[18] Navíc díky malé velikosti databáze (jejíž ERA model[1] je uveden na obrázku č. 10) KIVFS může databázový systém MySQL zkopírovat a spravovat celý obsah databáze v paměti serveru, což zvyšuje rychlost práce např. s již zmíněnými metadaty, jejichž obsah je často čten.



Obrázek 10: Původní ERA model databáze.

[1] V sekci 8 je uvedena jeho upravená a výsledná verze.

6.1 Přehled poskytovaných služeb

VFS server nabízí vyšším vrstvám adresářové a souborové služby. Jejich realizaci zajišťuje metadatový nebo souborový server. Poskytované služby jsou přehlednost uvedené v následující tabulce č. 3 (služby uvedené v závorkách jsou poskytovány automaticky v rámci celého úložiště).

Služby	Metadatový server	Souborový server
Souborové	otevření / vytvoření / zamknutí uzavření / odemknutí odstranění čtení atributů (správa jmenného prostoru)	čtení obsahu zápis dat (replikace)
Adresářové	vytvoření odstranění výpis obsahu	-

Tabulka 3: Poskytované služby

Z tabulky je patrné, že jmenný prostor (sekce 6.2.1) je spravován pouze metadatovým serverem. Díky tomu má každý KIVFS server kompletní informace o celém úložném prostoru.

6.2 Návrhy pro optimalizaci a rozšíření

V úvodu práce byly blíže popsány požadavky, které musí splňovat každé rozsáhlé datové úložiště, aby bylo v praxi použitelné. Na základě těchto požadavků a zmíněných cílových požadavků na vlastnosti KIVFS jsou v následujících bodech specifikované oblasti vhodné k optimalizaci:

- dostupnost a spolehlivost datového úložiště
  - efektivní přístup k souborům (sekce 6.2.3),
  - snížení omezování klientů vlivem současných vlastností KIVFS, mezi které patří např. poskytovaný jmenný prostor (sekce 6.2.1) či replikace (sekce 8.10),
  - podpora lokální cache a offline operací (sekce 6.2.5),
  - podpora pro obnovu dat (sekce 6.2.8),
- podpora mobilních platforem s ohledem na jejich omezení, mezi které patří např. omezená velikost lokální cache (sekce 6.2.3 a 6.2.5),
- zabezpečnost uložených dat (sekce 6.2.10),

- rozšiřitelnost a flexibilita datového úložiště (sekce 6.2.2).

V následujících podsekcích jsou tyto body dále rozebrány a jsou navrženy postupy, jak jich docílit.

6.2.1 Jmenný prostor

Nejvíce omezující vlastností současné implementace KIVFS z hlediska uživatelů je nemožnost sdílet jejich uložená data mezi sebou. Ta je zapříčiněna chybějící implementací mechanismu pro omezení přístupu k souborům, což vedlo k nutnosti poskytovat klientům privátní jmenný prostor. Toto je kritická vlastnost systému, kterou je potřeba upravit, aby se stal dostupnější pro běžné uživatele.

Jediným možným řešením je poskytovat uživatelům globální jmenný prostor a implementovat mechanismus zajišťující zabezpečení uživatelských dat formou omezení přístupu (viz sekce 6.2.10).

6.2.2 Efektivní využití služeb relačního databázového systému

Datové úložiště v KIVFS využívá relační databázový systém MySQL k ukládání všech informací o uživatelích, souborech, úložištích, apod. Pro přístup k němu používá tzv. konektor. Ten je dostupný v podobě sdílené knihovny pro programovací jazyk C, která poskytuje funkce pro správu databáze a dat v ní uložených.

Současná implementace metadatového serveru volá funkce konektoru přímo ve výkonných blocích zdrojového kódu, které zpracovávají požadavky od klientů a generují odpovědi na ně. To vede k nedostatečné flexibilitě a složitému použití jiného databázového systému z následujících důvodů:

- v případě potřeby změny použitého databázového systému je nutné provést úpravu velké části zdrojových kódů metadatového serveru,
- dochází k redundanci bloků zdrojového kódu, která mimo jiné zvyšuje jeho celkovou nepřehlednost a neefektivitu způsobenou nutností vykonávat stále stejnou sekvenci kroků:
  - generování SQL dotazu (či příkazu),
  - jeho vykonání (s využitím funkcí konektoru),
  - ošetření případných chyb (např. nedostupnost databázového systému),
  - zpracování a uložení dat z výsledku pro další použití.

Nedostatečnou flexibilitu v používání databázového systému lze odstranit vhodnou dekompozicí zdrojového kódu, která ve výsledku povede k implementaci metadatové mezivrstvy spravující veškerá uložená data v databázi. Toto řešení navíc přinese i následující výhody:

- nezávislost na použitém databázovém systému umožňující jeho jednoduchou změnu (i za databázový systém nevyužívající SQL),
- možnost využít jeho specifické funkce (např. podpora transakcí),
- přehlednost a zjednodušení zdrojového kódu metadatového serveru.

6.2.3 Model přístupu k souborům

Současná implementace přístupu k souborům je založena na modelu typu upload/download. Klientské aplikace ve svých aktuálních verzích pracují se souborem vždy jako s celkem, který je během úprav dostupný jako dočasná kopie nebo z lokální cache (pokud je na straně klienta implementována). Z tohoto důvodu použití upload/download modelu zatím nepředstavuje žádný problém a naopak jeho jednoduchost je výhodná - díky nenáročnosti na implementaci jak na straně sereru tak na straně klienta.

Již bylo zmíněno, že klíčovou a požadovanou vlastností KIVFS je podpora mobilních zařízení. Ty jsou limitované především v maximální velikosti kapacity lokální cache, která nikdy nebude dosahovat velikosti kapacity lokální cache klientské aplikace určené pro desktopovou platformu. Dalším omezením je citlivost na objem přenášených dat. Proto je nutné z hlediska zachování vysoké dostupnosti systému, upravit model přístupu k souborům tak, aby byl vhodný k použití i na mobilních platformách.

Řešením je implementace modelu vzdáleného přístupu k souborům, který využívá např. i OpenAFS. Ta kromě celkového zlepšení dostupnosti přinese konkrétně i následující výhody:

- možnost upravovat jen požadovanou část souboru (snížení nároků na datový přenos),
- nevyžaduje lokální cache na straně klienta.

Nevýhodou uvedeného řešení je vysoká režie na serveru v případě, že klient bude upravovat velký soubor po malých blocích a generovat velké množství požadavků, což vede k riziku zahlcení serveru a následné omezení služby. Tomu lze předejít např. implementací mechanismu, který bude změněné bloky souboru ukládat do vyrovnávací paměti a zapíše je až po dosáhnutí určité celkové velikosti nebo dokončení změn (uzavření souboru). Posílání souboru po velmi malých blocích ovlivňuje výslednou přenosovou rychlost. Vhodnou velikost posílaných bloků dat proto musí řešit klient.

6.2.4 Vyloučení souběžného přístupu k souboru

Metadatový server umožňuje díky své stavovosti řešit vzájemné vyloučení souběžného přístupu k souborům a konfliktních operací pomocí jednoduchých zámků. Kvůli tomu, že zámky nejsou odlišené zvlášť pro zápis a zvlášť pro čtení, nelze vykonávat vícenásobné čtení ze souboru.

Řešením problému je implementace zámků pro čtení a zápis (sekce 4.8.2).

6.2.5 Podpora klientské lokální cache a offline operací

Lokální cache na straně klientských aplikací zatím nebyla implementována. Z toho důvodu ji proto v současné verzi KIVFS nijak nepodporuje ani serverová část. Vzhledem k tomu, že využití lokální cache je v budoucnosti plánované pro zajištění dostupnosti dat i v případě nemožnosti navázat z nějakého důvodu spojení se serverem, podpora ze strany serveru je nutná.

Kvůli podpoře mobilních platforem, které jsou často omezeny velikostí lokálního datového úložiště, je nutné počítat i s omezenou maximální velikostí lokální cache. Kvůli tomu nelze uvažovat, že do lokální cache bude klient ukládat kopie všech souborů uživatele. Základním předpokladem pro zajištění podpory ze strany serveru je, že na straně klienta se:

- do lokální cache budou ukládat jen nejčastěji používané soubory,

nebo:

- z lokální cache budou odstraňovat nejméně používané soubory.

S implementací lokální cache na straně klienta je nutné navrhnout mechanismus, který zajistí konzistentní stav v ní uložených kopií souborů. Může totiž dojít ke konfliktní situaci, kdy klient má v lokální cache uloženou upravenou kopii souboru, který byl mezitím na straně serveru také upraven. Server proto nepovolí klientovi přepsat soubor, dokud nebude konflikt vyřešen.

Coda tento problém řeší verzováním jednotlivých souborů (sekce 4.7.1). Verzování souboru je realizováno čítačem, jehož hodnota odpovídá počtu změn souboru od jeho vytvoření. Případné vyřešení konfliktů v rozcházejících se verzích stejného souboru, který je uložen jak v lokální cache tak i na vzdáleném datovém úložišti, je přenecháno uživateli. Tento postup řešení pro zajištění konzistentního stavu bude použit i v KIVFS.

6.2.6 Replikace a konzistence uložených dat

Datové úložiště v KIVFS podporuje multimaster replikaci s využitím datacentrického modelu sekvenční kozistence. Všechny repliky souborů jsou přístupné pro čtení i zápis.

Současná implementace replikace způsobuje nedostupnost obsahu souboru po dokončení zápisu do něj. Důvodem je nutnost čekat, dokud nejsou provedeny změny ve všech replikách souboru. Soubor je proto stále uzamčen a jeho obsah nepřístupný, dokud nejsou všechny repliky aktualizovány a následně není odemčen a jeho obsah zpřístupněn uživatelům. To přináší následující nevýhody ovlivňující dostupnost celého KIVFS:

- okamžité zatížení všech souborových serverů spravujících repliku daného souboru,
- dojde-li během zápisu k chybě, je nutné aby byla zpracována všemi souborovými servery, což má za následek další zvýšení jejich režie a zátěže,
- obsah souboru není přístupný ihned po jeho nahrání.

Řešením je zachování multimaster replikace a implementace již zmíněného verzování souborů, resp. jejich replik. Kvůli zajištění stálé dostupnosti je nutné, aby replikace probíhala automaticky na pozadí[2]. Zápis proto bude prováděn pouze

[2]Narozdíl od OpenAFS (stabilní verze 1.6.1), u kterého je replikace, jež je typu master/slave, spouštěna „na vyžádání“ správcem.

do jedné repliky souboru. Po jeho dokončení dojde ke zvýšení její verze a ostatní repliky souboru přejdou do nekonzistentního stavu. Jejich postupnou aktualizací dojde k jejich navracení do konzistentního stavu a následnému zpřístupnění jejich obsahu uživatelům. Po aktualizaci všech replik souborů dojde ke znovuobnovení konzistentního stavu celého distribuovaného úložiště.

Sekvenční model konzistence dat z pohledu vyšších vrstev zůstane zachován díky zajištění přístupu pouze k replikám souborů s nejvyšší verzí. Navržené řešení přinese v kombinaci s výše zmíněnou implementací výlučného přístupu k souborům pomocí zámků pro čtení a zápis (sekce 6.2.4) následující výhody:

- snížení zatížení souborových serverů díky postupné synchronizaci zastaralých replik na pozadí s jejich aktuálními verzemi,
- zajištění dostupnosti obsahu alespoň jedné z replik souboru ihned po jeho nahrání.

6.2.7 Sémantika sdílení souboru

Sdílení souborů v rámci KIVFS je implementováno dle *unixové* sémantiky (sekce 4.8.1). Kvůli nutnosti okamžitě synchronizovat požadavky na zápis se zvyšuje režie a zatížení všech serverů. To z pohledu klienta vede ke zpoždění při čekání na zpracování požadavků a snižuje propustnost systému.

Řešením je implementace *relační* sémantiky (sekce 4.8.1), která je umožněna díky stavovosti metadatového serveru, a již zmíněného výlučného přístupu k souborům pomocí zámků pro čtení a zápis (sekce 6.2.4).

6.2.8 Obnova po výpadku

Současná verze KIVFS nemá implementovaný žádný mechanismus zajišťující obnovu dat po výpadku. To může vést ke snížení stability celého systému.

Proces obnovy dat po výpadku datového úložiště musí mít na starosti synchronizační vrstva, která je jako jediná schopná rozpoznat, že k výpadku došlo, a může zahájit obnovu dat. Ta z pohledu serveru spočívá ve zpracování nezpracovaných požadavků, které byly ve zbytku systému (tj. na všech ostatních serverech) již zpracovány.

Mechanismus pro obnovu dat po výpadku vyžaduje ukládat všechny požadavky na změnu dat a mít tak dostupný seznam provedených operací (požadavků) v rámci každého datového úložiště. To lze v rámci datové vrstvy zajistit implementací podpory pro logování požadavků synchronizační vrstvou.

6.2.9 Správa uložených dat a jejich deduplikace

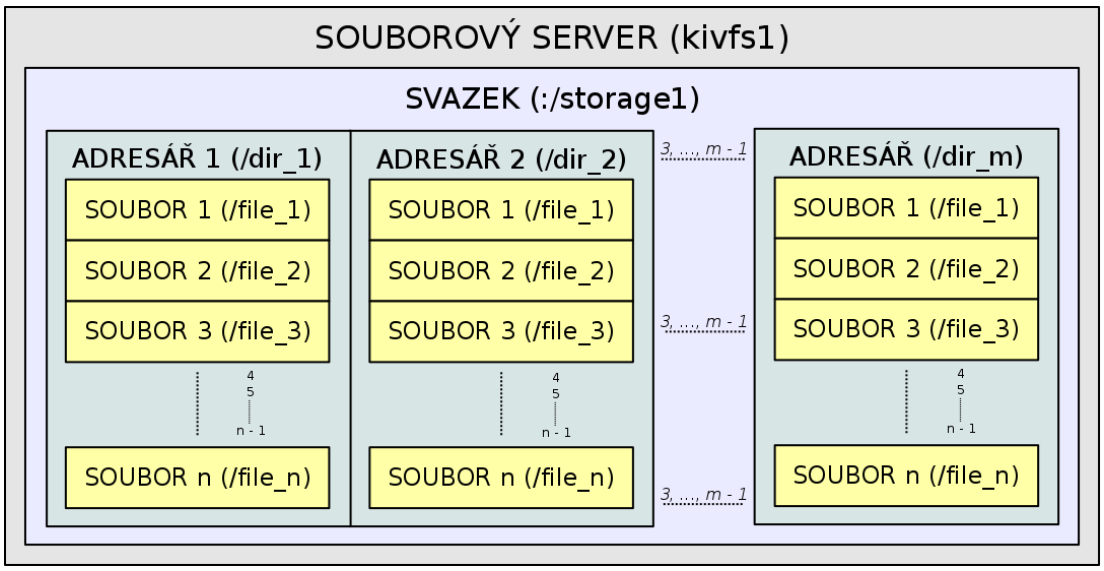
Soubory uložené na svazcích souborových serverů jsou organizováné dle již dříve navržené struktury pro ukládání dat.[18] Její schéma je znázorněno na obrázku č. 11 a spočívá v rozmístění celých souborů v určitém počtu do jednotlivých adresářů, jejichž počet není omezen. Adresáře se dynamicky vytvářejí a zaplňují v závislosti na aktuálním počtu souborů v nich uložených. Počet souborů v adresáři je omezen kvůli možnosti optimalizovat strukturu pro ukládání dat pro použité souborové

systémy, jejichž výkonnost je přímo ovlivněna celkovým počtem souborů v pracovním adresáři. Mapování uložených souborů v této struktuře zajišťuje metadatový server, který spravuje jmenný prostor a poskytuje souborové služby (tabulka č. 3).

V rámci celého systému lze optimalizací struktury pro ukládání dat, která spočívá v rozdělování uložených souborů menších částí (dále označovaných jako podsoubory), značně ovlivnit jeho:

Výkonnost. Místo replikace celého souboru je možné provádět replikaci jen jeho změněných podsouborů.

Rozšiřitelnost. Rozdělení souborů do menších částí odstraní limit pro maximální velikost souboru a tím i další snížení závislostí na použitých technologiích, která se v tomto případě týká vlastností použitého souborového systému na daném svazku souborového serveru.



Obrázek 11: Schéma struktury pro ukládání dat.

V souvislosti s replikací se nabízí možnost její optimalizace spočívající v synchronizaci pouze změněných podsouborů dané repliky souboru. K tomu je nutné navrhnout a implementovat mechanismus, který rozpozná změněné relevantní podsoubory. Ten může být založen porovnávání kontrolních součtů jednotlivých podsouborů s využitím algoritmů MD5[14], SHA[15], apod.

Na základě hledání shody mezi kontrolními součty lze postavit i experimentální deduplikaci uložených podsouborů, která se dnes zatím u žádného běžně používaného DFS nevyskytuje. Jejím cílem je odstranění redundance uložených podsouborů v rámci datového úložiště jejich náhradou za odkazy na jeden zdrojový podsoubor. V případě, že uživatel chce zapsat do deduplikovaného podsouboru (tj. odkazu na zdrojový podsoubor), je nutné zajistit jeho zpětnou duplikaci ještě před provedením samotného zápisu. Za jednoznačné přínosy deduplikace lze uvažovat:

- optické zvýšení kapacity datového úložiště,
- nižší režie datového úložiště.

V rámci KIVFS lze deduplikaci uložených podsouborů řešit pouze na úrovni jednotlivých svazků souborových serverů.[18] Deduplikovat uložená data v rámci celého souborového serveru (resp. všech jeho svazků) přináší riziko nedostupnosti deduplikovaných dat. To může nastat během selhání svazku obsahujícícho zdrojový podsoubor, na který odkazují deduplikované podsoubory. Teoreticky možná je i deduplikace dat v rámci celého KIVFS. Její implementace je ale bezúčelná, protože by docházelo i k deduplikaci replik.

6.2.10 Zabezpečenost uložených dat

V rámci KIVFS nejsou implementovány žádné mechanismy řešící zabezpečení uložených dat a jejich sdílení. Uživatelům je dispozici privátní jmenný prostor a kvůli tomu mají přístup pouze k obsahu souboru, jehož jsou vlastníkem. Nemožnost sdílet data dalším uživatelům pro ně značně snižuje použitelnost celého systému.

Zpřístupnění všech souborů a tedy umožnění jejich sdílení mezi všemi uživateli lze docílit pouze implementací globálního jmenného prostoru (sekce 4.3.1). Ten přináší bezpečností rizika, mezi která patří:

- nevyžádaná změna obsahu souboru,
- nechtěné poskytnutí obsahu souboru,
- smazání celého souboru,

třetí stranou.

Uvedená rizika lze odstranit pouze implementací mechanismu, který bude umožňovat nastavit omezený přístup k souborům a následně dohlížet na jeho respektování. V rámci KIVFS bude omezený přístup k souborům realizován prostřednictvím seznamu přístupových práv (sekce 4.9), který využívají i např. OpenAFS či Coda.

Dalším rizikem, které se týká bezpečnosti uložených dat, je možnost fyzického útoku na datové úložiště. Utočníkovým cílem je v tomto případě získání přímého přístupu k datům, která jsou uložená na svazcích napadaného souborového serveru. Pokud uživatel nezašifroval data před jejich nahráním prostředníctvím klientské aplikace na napadený souborový server, umožnil útočníkovi jednoduchý přístup.

Jedinou účinnou ochranou proti fyzickému útoku na souborový server a následnému zneužití uložených dat je prevence spočívající v implementaci symetrického šifrování bloků přijatých dat před jejich samotným uložením. Asymetrické šifrování není možné z důvodu potřeby uchovávat různé klíče na souborových serverech a nutnosti mít ve výsledku na jejich svazcích uložená data o stejné velikosti.

6.2.11 Kvóty virtuálních svazků

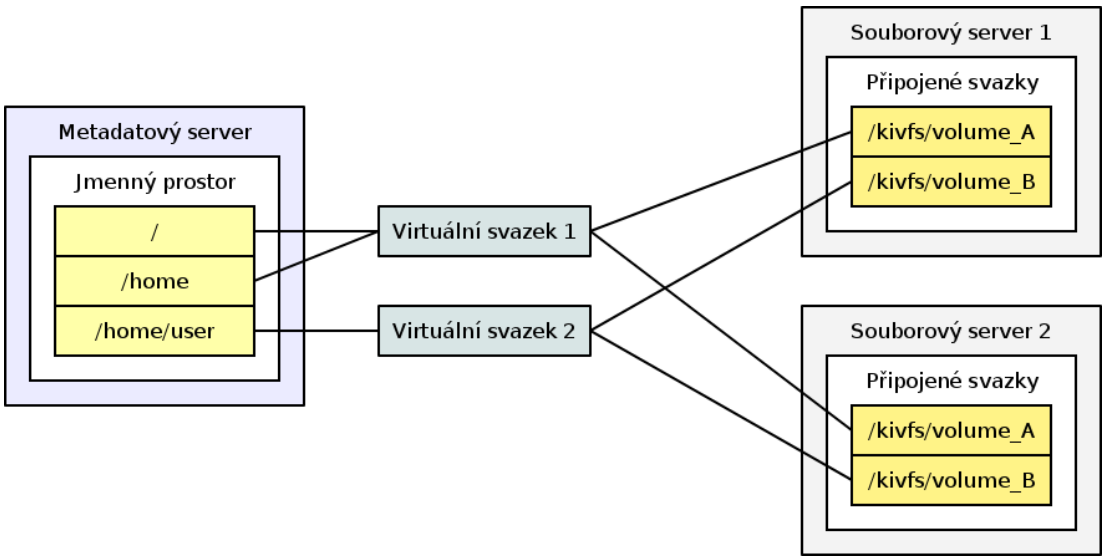
Datové úložiště KIVFS neumožňuje nastavit žádné omezení uživatele z hlediska dostupné kapacity úložiště. Uživatelé se proto mohou bez špatného úmyslu pokoušet

nahrát soubory, jejichž velikost je větší než celková kapacita datového úložiště, která je rovna celkové kapacitě svazků souborových serverů. V důsledku toho může na souborovém serveru teoreticky dojít k následujícím situacím, které omezují dostupnost a spolehlivost celého systému:

- nemožnost přijmout a uložit soubor, který se klient pokouší nahravát,
- neočekávaným chybám souborového systému použitého na svazku, připojeném k souborovému serveru, které mohou vést až ke ztrátě dříve uložených dat.

Uvedeným situacím lze předejít implementací mechanismu, který umožní nastavit diskovou kvótu omezující dostupnou kapacitu tzv. virtuálních svazků.[18] Ty jsou v rámci jménného prostoru použité jako přípojné body adresářů k jednotlivým svazkům souborových serverů. Příklad jejich použití a mapování na úrovni serverů je znázorněn na obrázku č. 12, ve kterém platí, že:

- svazky *A* a *B* jsou replikovány na oba souborové servery
- *kořenový* adresář a adresář *home* jsou připojeny k replikovanému svazku *volume_A*
- domovský adresář *user* je připojen k replikovanému svazku *volume_B*



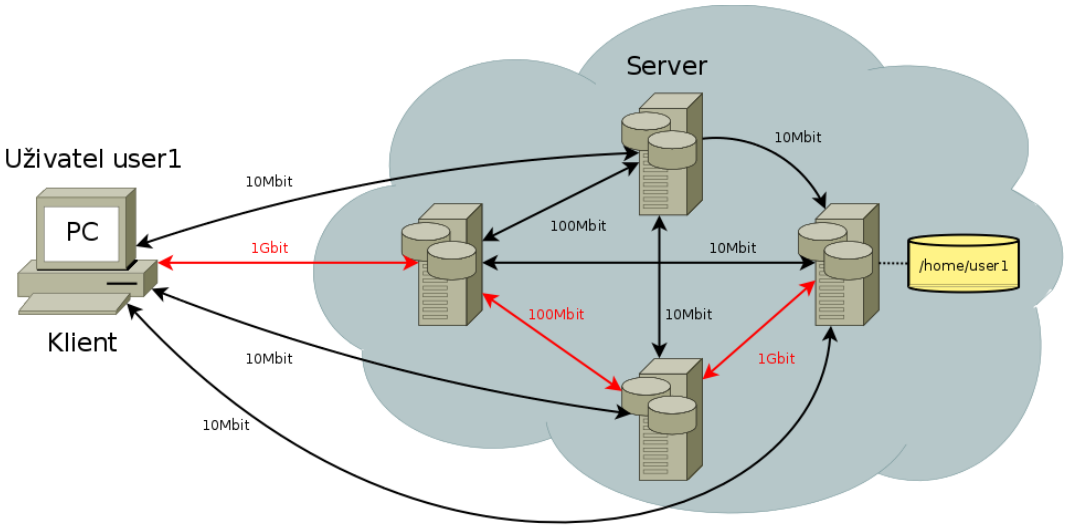
Obrázek 12: Virtuální svazky a jejich mapování na úrovni serverů.

Uživatele *user* tak lze z jeho pohledu omezit diskovou kvótou přidělenou např. k jeho domácímu adresáři, která omezí dostupnou kapacitu adresáře.

6.2.12 Tunelování přístupu k souborům

Klient je vždy během přenosu souboru připojen k souborovému serveru, který je pro něj z hlediska jeho dostupnosti nejvhodnější a obsahuje aktuální repliku požadovaného souboru. I tak ale může být navázané spojení uskutečněno s geograficky vzdáleným serverem a rychlost přenosu souboru může být omezena.

Za předpokladu, že servery budou vždy vzájemně propojené rychlou a stabilní linkou, může být tento problém vyřešen vytvořením tunelovaného spojení mezi nejbližším souborovým serverem a dalším souborovým serverem, který spravuje požadovanou repliku souboru. Klient se proto může stále připojovat jen k nejbližšímu souborovému serveru, který nemusí obsahovat požadovanou repliku souboru, ale skrze tunel bude její obsah transparentně poskytovat z jiného souborového serveru. Rychlost přenosu souboru z pohledu klienta tak může být značně zvýšena. Na obrázku č. 13 je uveden příklad, ve kterém uživatel přistupuje ke svému svazku nejvyšší dostupnou rychlostí skrze 100Mbit tunel.



Obrázek 13: Tunelování spojení (100MBbit tunel vyznačen červěně).

7 Sdílená knihovna libkivfscore

Během paralelně probíhajícího vývoje jednotlivých vrstev serveru byly v jejich zdrojových kódech postupně identifikovány bloky, které zbytečně implementují podobnou funkcinalitu. Zpočátku se jednalo o funkce realizující síťovou komunikaci. K těm zanedlouho přibyly další funkce, které řešily jednotné formátování výpisovaných informací na obrazovku či do souboru, správu vlastních datových struktur, apod.

Duplicitní části zdrojového kódu měly negativní vliv na jeho přehlednost, která vedla k častějšímu výskytu programátorských chyb. V rámci potřeby odstranit duplicitní funkce, zajistit dodržování programovacích konvencí a celkově zpřehledněnit zdrojové kódy KIVFS vznikla sdílená knihovna *libkivfscore*, jejímž cílem hlavním je usnadnit další vývoj.

V průběhu dalšího vývoje se sdílená knihovna *libkivfscore* stala základním stavebním kamenem celého KIVFS, který využívají i klientské aplikace a nebo z něj přinejmenším vycházejí. Základní funkce[3], které poskytuje, lze rozdělit do následujících kategorií:

Síťová komunikace (core/kivfs-net.h). Funkce slouží jako nadstavba síťové komunikace skrze sokety s využitím KIVFS protokolu. Umožňuje vytvářet zprávy typu požadavek či odpověď a poskytuje funkce pro jejich spolehlivou výměnu.

Správa datových typů (core/kivfs-structs.h). Funkce poskytují jednotnou správu abstraktních datových typů a struktur včetně jejich serializace do pole bajtů (a deserializaci).

Správa vláken (core/kivfs-thread.h). Funkce slouží ke správě seznamu POSIX vláken a rozšiřují možnosti jejich použití.

Logování (core/kivfs-logger.h). Funkce definují jednotný formát pro tisknutí informací na obrazovku či do souboru a umožňují využívat systémový log.

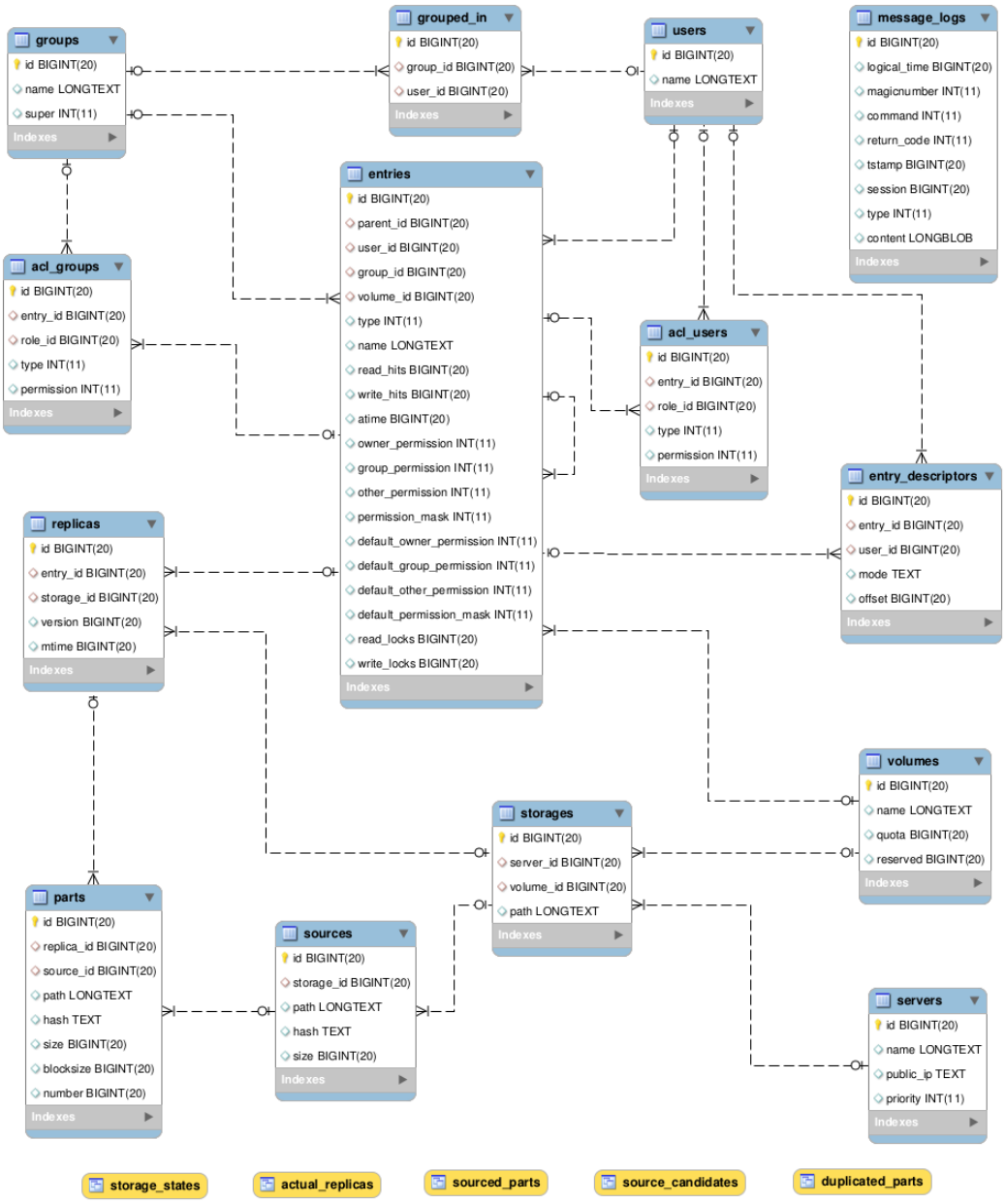
Správu sezení (core/kivfs-session.h). Funkce umožňují spravovat sezení (tzv. session). Jde především o správu všech navázaných spojení, zajištění bezpečného přístupu do prostoru paměti, který je sdílen mezi všemi sezeními.

Popis chyb (core/kivfs-constants.h). Funkce zajišťují popis všech chyb, které se mohou v rámci systému vyskytnout.

[3]Konkrétní popis funkcí lze nalézt v dokumentaci, která je dostupná na přiloženém CD.

8 Optimalizace datového úložiště KIVFS

Během implementace navržených optimalizací a rozšíření bylo nutné průběžně zasahovat do původního schématu použité relační databáze. Kvůli zjednodušení všechny dále popisované řešení počítají s již upraveným a připraveným schématem databáze. Následuje proto jeho ERA model, který je uveden na obrázku č. 14, a stručný popis změn.



Obrázek 14: ERA model současné databáze.

Tabulka entries

- vznikla sloučením tabulek files a directories,
- obsahuje informace o všech souborech,
- přidány atributy umožňující nastavení:
  - základních přístupových práv k souboru (pro vlastníka, skupinu a ostatní),
  - hodnot čítačů přístupů k souboru (pro čtení, zápis),
  - hodnot čítačů zámků souboru (pro čtení, zápis),
  - typu souboru (běžný soubor, adresář).

Tabulka volumes

- obsahuje informace o virtuálních svazcích,
- přidány atributy, které umožňují nastavení:
  - kapacity svazku,
  - rezervovaného místa.

Tabulka replicas

- obsahuje informace o replikách souborů,
- přidány atributy, které umožňují nastavení:
  - verze repliky,
  - času poslední modifikace repliky,

obsahující informace o replikách souborů.

Tabulka groups

- obsahuje informace o všech skupinách uživatelů,
- přidány atribut označující skupinu administrátorů.

Tabulka acl_users

- přidána z důvodu nutnosti efektivně uchovávat informace o (jmenném) seznamu uživatelů, kteří mají přidělen přístup k souboru.

Tabulka acl_groups

- přidána ze stejného důvodu jako tabulka acl_users, ale pro skupiny.

Tabulka sources

- přidána kvůli nutnosti uchovávat a spravovat informace o deduplikovaných podsouborech.

Tabulka parts

- přidána kvůli nutnosti uchovávat informace o podsouborech.

Tabulka entry_descriptors

- přidána kvůli nutnosti uchovávat a spravovat informace o otevřených souborech.

Tabulka message_logs

- přidána kvůli nutnosti uchovávat a spravovat požadavky, na základě kterých lze provést obnovu systému.

Z potřeby rychlého přístupu k souhrným informacím z více tabulek byly vytvořeny následující pohledy:

Pohled actual_replicas

- obsahuje informace o aktuálních replikách souboru.

Pohled storage_states

- obsahuje informace o aktuálním stavu všech svazků.

Pohled source_candidates

- obsahuje seznam všech duplicitních podsouborů, ze kterých lze vytvořit zdrojové podsoubory.

Pohled duplicated_parts

- obsahuje seznam všech duplicitních podsouborů, které jsou určené k deduplikaci.

Pohled deduplicated_parts

- obsahuje seznam všech deduplikovaných podsouborů.

8.1 Metadatová mezivrstva

Implementovaná metadatová mezivrstva slouží jako rozhraní mezi databázovým systémem a metadatovým serverem. Jejím cílem je spravovat a hlavně poskytovat data uložená v relační databázi s využitím abstrakních datových typů a struktur, které jsou definovány ve sdílené knihovně *libkivfscore*, prostřednictvím těchto základních operací:

- výběr záznamů z databáze,
- vložení nového záznamu do databáze,

- smazání záznamů z databáze,
- úprava záznamů v databázi.

Jako příklad lze uvést operaci pro výběr souboru z databáze, který je realizován voláním níže uvedené funkce. Seznam všech funkcí pro správu souboru v databázi, je uveden v příloze A.

```
 /* vybere soubor v adresáři s daným názvem */
int db_get_file(db_transaction_t *transaction,  // identifikátor transakce
                kivfs_entry_t **p_entry,        // vybraný soubor
                kivfs_entry_t *parent_entry,    // rodičovský adresář
                char *name);                    // název souboru
```

V tělech těchto funkcí se generuje SQL příkaz, který se předává vlastnímu databázovému konektoru k vykonání. Ten v případě úspěšného vykonání příkazu vrátí jeho výsledek, který je dále zpracován a hodnoty z něj uloženy do datových struktur.

8.1.1 Vlastní databázový konektor

Pro přístup k datům uloženým v databázi se využívají funkce, které jsou poskytovány konektorem k MySQL. Jeho použití značně omezuje chybějící podpora sdílení navázaného spojení k databázovému systému (tzv. poolování) mezi více vlákny. Tento problém je vyřešen použitím vlastního konektoru k databázovému systému, který v současné verzi implementován jako nadstavba MySQL konektoru.

Při inicializaci konektoru dojde k navázaní určitého počtu spojení k databázovému systému, který je dán parametrem *pooling*[4] v konfiguračním souboru serveru. Všechna spojení jsou poté uložena do spojového seznamu sdíleného mezi všemi vlákny.

V případě, že vlákno potřebuje přistupovat k datům uloženým v databázi, je vybráno spojení, které je právě volné a není rezervováno žádným dalším vláknem. Toto spojení je následně označeno jako rezervované a vlákno může skrze něj začít vykonávát požadované operace s daty. Po dokončení všech operací vlákno zruší rezervaci spojení a to je opět označeno jako volné.

Požaduje-li vlákno přístup k datům uloženým v databázi a všechna spojení jsou rezervovaná, pak zaloguje varovnou zprávu:

```
Dababase connection limit has been reached. Please increase the pool limit in
cofiguration file.
```

a následně musí čekat, dokud u některého z nich nebude rezervace zrušena a nebude uvolněno. Důvodem pro omezený počet spojení k databázovému serveru je ochrana proti DDoS útoku.

Navázání a uknočení spojení lze realizovat prostřednictvím volání níže uvedených funkcí. Seznam všech dalších funkcí, které databázový konektor poskytuje, je uveden v příloze B.

[4]Výchozí hodnota je 100 - tento počet spojení se během testování výkonnosti datového úložiště projevil jako dostačující.

```
/* naváže spojení s databázovým systémem */
int db_connect(char *host,                // IP adresa nebo hostname serveru
               uint16 port,               // port serveru
               char *user,                // uživateslé jméno
               char *pass,                // uživatelské heslo
               char *database,            // název (schématu) databáze
               unsigned int pool_size);   // max. počet spojení

/* ukončí spojení s databázový systémem */
void db_disconnect();
```

Použití vlastního databázového konektoru umožňuje jednoduchou změnu databázového systému. Té lze docílit úpravou těl výše uvedených funkcí. Experimentálně byly implementovány databázové konektory pro relační databázové systémy Oracle a PostgreSql. V jejich vývoji se zatím dále nepokračuje.

8.1.2 Mapování relací na datové typy

Jednotlivým relacím ve schématu databáze odpovídají datové struktury, ke kterým jsou k dispozici funkce pro správu. Jako příklad lze uvést datové struktury a funkce odpovídající relaci mezi soubory a ACL, který je kvůli své délce v příloze C.

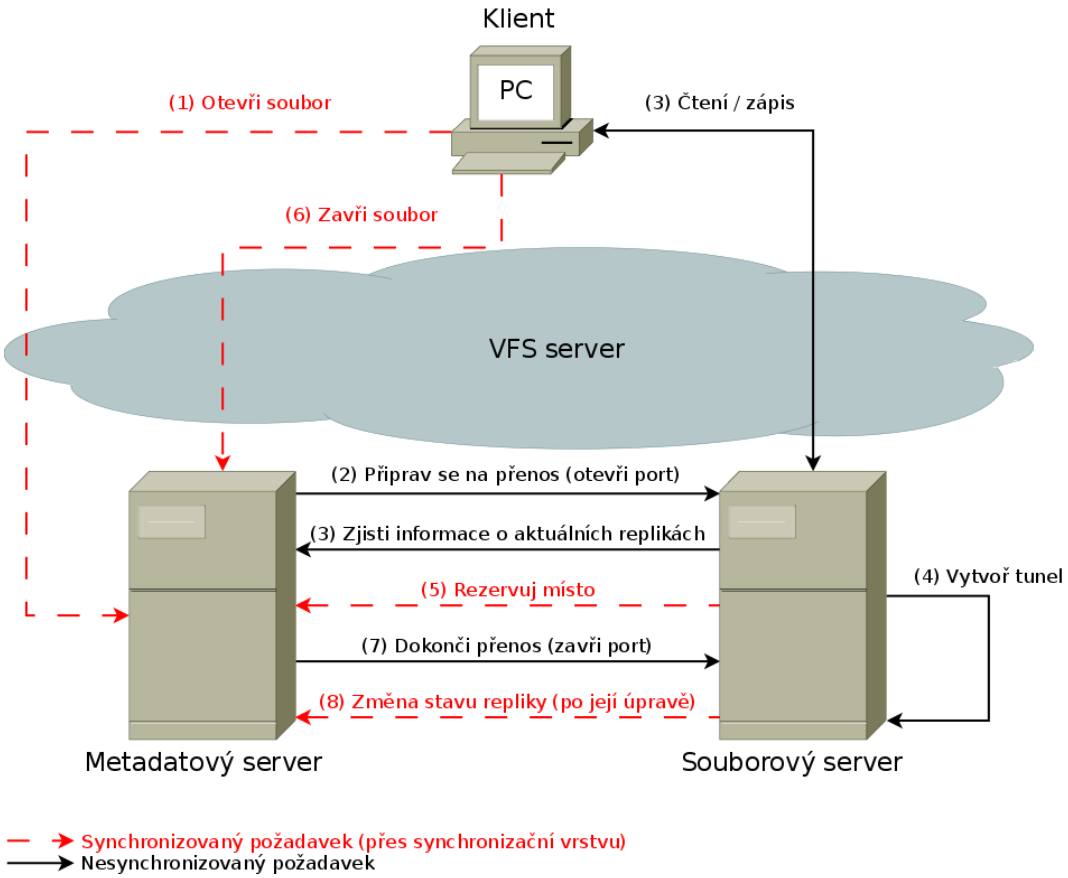
8.2 Vzdálený přístup k souborům a relační sémantika jejich sdílení

Největší změna řešící optimalizaci datového úložiště byla provedena v rámci úpravy přístupu k souborům. Model upload/download byl nahrazen modelem vzdáleného přístupu k souboru a byla implementována relační sémantika sdílení souborů (změny v souboru jsou pro ostatní uživatele viditelné až po jeho zavření). V zásadě šlo o přidání následujících možností pro práci se souborem:

- otevřít soubor v režimu pro:
  - čtení,
  - zápis,
  - zápis na konec souboru,
  - čtení a zápis,

  a vytvořit jeho popisovač (tzv. file descriptor),
- možnost nastavovat aktuální pozici v souboru,
- uzavřít soubor pomocí jeho popisovače.

Výsledné řešení je popsáno formou jednotlivých kroků, které se odehravají na straně serveru během přenosu obsahu souboru mezi ním a klientem. Na obrázku č. 15 jsou pro snažší představu tyto kroky znázorněny.



Obrázek 15: Přenos souboru.

8.2.1 Otevření souboru

Požadavek na otevření souboru zpracovává metadatový server. Ten po jeho přijetí zjistí, který soubor má být otevřen a v jakém režimu. Seznam a popis jednotlivých režimů je uveden v tabulce č. 4.

Poté ověří, zda je možné soubor v daném režimu otevřít ověřením následujících podmínek:

1. uživatel má k souboru umožněný přístup s ohledem na dostupný seznam přístupových práv (způsob ověření přístupu k souboru je popsán v sekci 8.4),

2. soubor je odemčen (způsob ověření zámků souborů je popsán v sekci 8.5).

Pokud soubor neexistuje a je otevřen v režimu pro zápis, je nutné jej vytvořit. Do tabulky *entries* je vložen nový záznam reprezentující soubor. V závislosti na rodičovském adresáři a k němu připojenému virtuálnímu svazku (záznamu z tabulky *volumes*), který slouží jako přípojný bod k jednotlivým svazkům souborových serverů (záznamy z tabulky *servers*), zjistí výsledné fyzické umístění replik souboru (záznamy z tabulky *storages*). Pro každý ze zjištěných svazků souborových serverů, se v tabulce *replicas* vytvoří záznamy odpovídající zatím prázdným replikám souboru. Jejich verze je nastavena na výchozí hodnotu jedna.

Režim	Popis
Zápis	V případě, že soubor neexistuje, je vytvořen. Obsah existujícícho souboru je vynulován.
Zápis na konec souboru	V případě, že soubor neexistuje, je vytvořen. Aktuální pozice v souboru je nastavena na jeho konec.
Čtení a zápis	V případě, že soubor neexistuje, je vytvořen.
Čtení	Soubor musí existovat.

Tabulka 4: Režimy pro otevření souboru.

Otevření souboru je provedeno vložením nového záznamu do tabulky *file_descriptors*, po kterém následuje požádání souborového serveru o připravení se na přenos jeho obsahu. Souborový server otevře náhodný port, který je určen pouze pro přenos právě otevřeného souboru.

Na závěr tohoto kroku metadatový server odešle odpověď klientovi na požadavek otevření souboru. Ta obsahuje informace o otevřeném souboru, IP adrese souborového serveru připraveného na přenos souboru a portu, na kterém poslouchá.

8.2.2 Příprava na přenos obsahu souboru

Přenos obsahu souboru začíná připojením klienta k souborovému serveru. Ten okamžitě po navázání spojení zjistí od metadatového serveru souhrné informace o všech aktuálních replikách souboru z pohledu *actual_replicas*, které jsou dostupné, a také souborových serverech, jež je spravují.

Je-li jedním z nich, pak opět od metadatového serveru zjistí dostupné informace o:

- všech podsouborech, které tvoří obsah repliky (informace z tabulky *parts*),
- svazku, na kterém jsou podsoubory uložené (informace z tabulky *storages*),

a je připraven na záhájení přenosu obsahu repliky souboru.

V případě, že repliku souboru sám nespravuje, musí vytvořit tunel skrze souborové servery mezi sebou samým a tím, který ji spravuje. Poté veškerá jeho činnost sestává z přeposílání požadavku do vytvořeného tunelu. Pro souborový server na druhé straně tunelu se souborový server, ke kterému je připojen klient, sám stává klientem. Vytváření tunelů je popsáno v sekci 8.9.

8.2.3 Přenos obsahu souboru

Kvůli zjednodušení je z dalšího textu vynechán popis manipulace s podsoubory repliky, která je zvlášť popsána v sekci 8.3. Je uvažováno, že replika je tvořena jen jedním souborem (podsouborem).

Jakmile je souborový server připraven k přenosu obsahu repliky souboru, může začít přijímat požadavky na čtení nebo zápis klienta. Po každém obdržení požadavku musí z bezpečnostních důvodů zjistit, zda je určen pro původně otevřenou repliku souboru a také jestli aktuální operace odpovídá jeho režimu otevření.

Jde-li o požadavek, který zapisuje do repliky souboru, pak vyhodnotí, zda je nutné rezervovat místo pro nová data a případně se o to pokusit. K tomu dojde jen tehdy, je-li zapisováno na konec repliky souboru[5]. Rezervace místa je popsáná v sekci 8.8.1.

S pomocí metadového serveru zjistí aktuálně nastavenou pozici v replice souboru. Pokud odpovídá pozici po poslední provedené operaci, může pokračovat v práci s aktuálně otevřenou replikou souboru. Jestliže byla pozice klientem změněna nebo zatím nebyla otevřena žádná replika souboru, pak musí najít a otevřít příslušnou repliku souboru. Tento mechanismus zajišťuje minimální režii manipulace se soubory, jejichž obsah může být zašifrován (sekce 8.7), a zvyšuje tak odolnost serveru vůči nadměrnému zatížení, ke kterému by jinak mohlo dojít v případě častého provádění operací čtení či zápisu po malých blocích.

V tuto chvíli začíná přenos obsahu otevřené repliky souboru. Klient vždy posílá požadavek, ve kterém je uvedeno, kolik dat chce přečíst či zapsat. V případě operace pro čtení, server jen čte přímo z otevřené repliky souboru a posílá jeho obsah. V případě zápisu je před samotným otevřením repliky souboru vytvořena její dočasná kopie, do které je následně zapisováno. Po úspěšném dokončení zápisu je původní replika souboru nahrazena její kopií. Tento přístup zajišťuje, že v případě výskytu jakékoliv chyby (např. výpadek serveru) je zajištěn konzistentní stav původní repliky souboru.

Není-li přenos dokončen, server zopakuje předchozí krokem - zjistí poslední nastavenou pozici v replice souboru a připraví se na další přenos.

8.2.4 Dokončení přenosu obsahu souboru

Ukončení přenosu obsahu souboru je detekováno obdržením informace od metadatového serveru, že soubor byl zavřen. Došlo-li během přenosu ke změně repliky souboru, pak souborový server zvýší její verzi a odešle zprávu synchronizační vrstvě obsahující:

- jednoznačný identifikátor repliky,
- seznam podsouborů tvořících obsah repliky,
- verze repliky.

Synchronizační vrstva zajistí propagaci zprávy na všech metadatových serverech, jež zaregistrují aktualizaci repliky a příslušných podsouborů upravením odpovídajících záznamů v tabulkách *replicas* a *parts*. Z hlediska souborového serveru je v tuto chvíli přenos souboru dokončen a zavírá port.

8.2.5 Uzavření souboru

Požadavek na zavření souboru zpracovává opět metadatový server. Po jeho přijetí zjistí z tabulky *file_descriptors*:

[5]K rezervaci místa dojde i v případě zapisování do repliky právě vytvořeného souboru.

- který soubor má být zavřen,
- v jakém režimu byl soubor otevřen.

V závislosti na získaných informacích odstraní příslušný záznam z tabulky *file_descriptors* a upraví stav zámků souboru v tabulce *entries*. Následně oznámí souborovému serveru, že soubor byl zavřen a jeho přenos má být ukončen. Posledním krokem je odeslání odpovědi klientovi, že soubor byl zavřen. V tuto chvíli je celý přenos souboru úspěšně ukončen.

Pokud byla během přenosu souboru změněna jeho replika, ostatní jeho repliky jsou v nekonzistentním stavu. Návrat zpět ke konzistentnímu stavu prostředníctvím replikace je popsán v sekci 8.10.

8.3 Rozdělování souboru do podsouborů

Souborový server během zápisu do repliky souboru průběžně kontroluje její velikost. V případě, že dosáhne určité velikosti bloku dat, který je definován v konfiguračním souboru jako hodnota parametru *blocksize*[6], pak soubor uzavře a začne zapisovat do dalšího souboru. Replika je ve výsledku tvořena z jednoho a více podsouborů.

Podsoubory jsou organizované dle navržené struktury pro ukládání dat[18]. Počet souborů v jednom adresáři je definován v konfiguračním souboru jako hodnota parametru *files_count*[7].

Souborový server odesílá po každém zápisu seznam všech vytvořených podsouborů tvořících obsah repliky souboru metadatovému serveru, který je eviduje v tabulce *parts*.

8.4 Seznam přístupových práv

Zabezpečení omezující přístup ke sdíleným souborům bylo v rámci metadatového serveru implementováno pomocí ověřování přístupu v závislosti na seznamu přístupových práv (ACL), které vychází z POSIX standardu.[24] Tabulka č. 5 obsahuje přehled dostupných přístupových práv.

Každý soubor (adresář) z tabulky entries má pevně definovaná práva pro:

- vlastníka,
- vlastnickou skupinu,
- ostatní,
- masku práv[8],

[6]Výchozí hodnota je 1GB a byla zvolena na základě provedených měření v mé bakalářské práci.[18]

[7]Výchozí hodnota je 5000 souborů v jednom adresáři a byla zvolena na základě provedených měření v mé bakalářské práci.[18]

[8]Maska práv omezuje existující práva pro vlastnickou skupinu a ostatní uživatele (narozdíl od souborové masky).

včetně jejich výchozích variant. Ke každému souboru je navíc možné přiřadit zvlášť definovaná práva pro roli (nebo výchozí roli), která může být typu uživatel či skupina. V rámci databáze jsou tyto dodatečně nastavená přístupová práva spravována v tabulkách *acl_users* a *acl_groups*.

Práva u nově vytvořených souborů či adresářů jsou určena dle výchozích práv jejich nadřazeného adresáře. Z práv nově vytvořeného souboru je automaticky odebráno právo na spouštění.

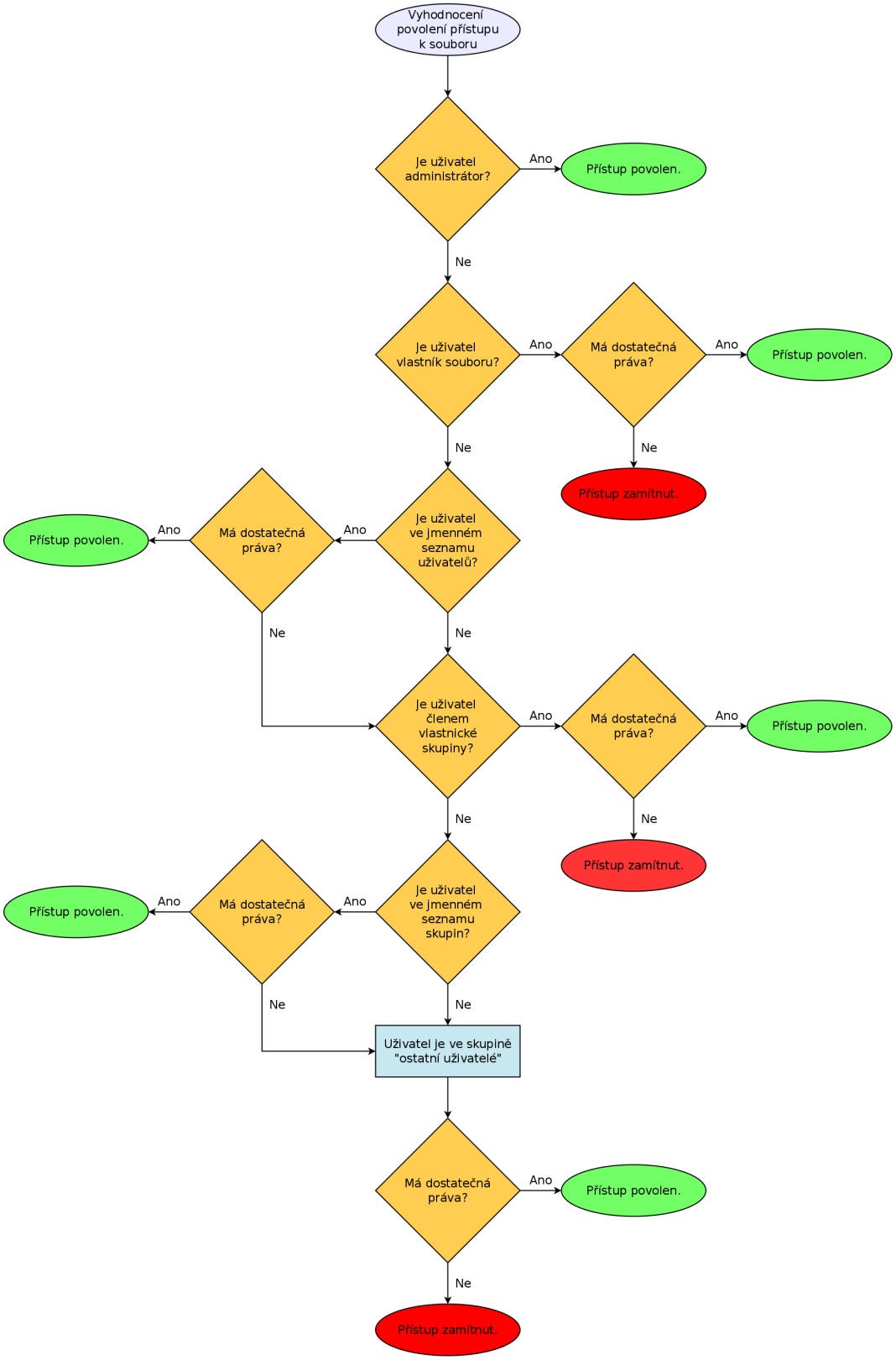
Výchozí práva pro kořenový adresář a všechny v něm vytvořené adresáře jsou 755 a maska 7. U nově vytvořených souborů jsou nastavaná práva 644 a maska 7. Výchozí práva u nově vytvořených adresářů jsou zděděna od jejich nadřazeného adresáře. Výchozí práva u nově vytvořených souborů se nedědí.

Osmičkově	**Čtení**	**Zápis**	**Spuštění**
0	ne	ne	ne
1	ne	ne	ano
2	ne	ano	ne
3	ne	ano	ano
4	ano	ne	ne
5	ano	ne	ano
6	ano	ano	ne
7	ano	ano	ano

Tabulka 5: Přístupová práva k souborům

8.4.1 Vyhodnocení omezeného přístupu k souboru

Metadatový server vyhodnocuje, zda má uživatel povolený přístup k souboru v závislosti na přiřazených ACL záznamech dle algoritmu znázorněného pomocí vývojového diagramu na obrázku č. 16.



Obrázek 16: Vyhodnocení povolení přístupu k souboru s využitím ACL.

8.4.2 Správa seznamu přístupových práv

Je-li uživatel vlastník souboru nebo je začleněn do skupiny admistrátorů, má možnost pomocí příkazů (požadavků), které jsou uvedeny v tabulce č. 6, měnit atributy souboru, jež se týkají přístupových práv k němu.

Příkaz KIVFS	**Parametry**	**Kód**	**Formát dat**	**Popis**
chown	uživatel soubor	45	%s %s	Nastaví vlastníka.
chgroup	skupin soubor	46	%s %s	Nastaví vlastnickou skupinu.
chmod	maska práva soubor	47	%d %d%d%d %s	Nastaví přístupová práva (vlastníka, vlastnické skupiny, ostatních) a jejich masku.
dchmod	maska práva soubor	48	%d %d%d%d %s	Nastaví výchozí přístupová práva (vlastníka, vlastnické skupiny, ostatních) a jejich masku.
setfacl	typ role práva soubor	49	%d %d %d %s	Nastaví přístupová práva pro danou roli (vytvoří či upravý jmenný záznam).
getfacl	soubor	50	%s	Získá ACL přiřazené k souboru.

Tabulka 6: Příkazy (požadavky) pro správu přístupová práv souborů

8.5 Úprava vzájemného vyloučení přístupu k souborům

Každý soubor lze zamknout pro čtení nebo zápis. Zámky jsou implementované jako čítače, jejichž hodnota odpovídá jejich aktuálnímu stavu. Kvůli zajištění stavovosti serveru jsou hodnoty zámku ukládány do tabulky *entries* a to konkrétně do hodnot sloupců *read_locks* a *write_locks*.

Při každém přístupu k obsahu souboru je ověřen stav jeho zámků. Dle režimu, ve kterém je potřeba soubor otevřít, se dle následujících pravidel vyhodnotí, zda je možné soubor zamknout:

Pro čtení. Soubor lze otevřít pro čtení, je-li hodnota čítače zámků pro zápis nulová. Pokus o zamknutí souboru pro čtení odpovídá následujícímu zjednodušenému pseudokódu:

```
kivfs_entry_t entry;

... /* inicializace proměnné entry pomocí
     * hodnot získaných z databáze */
```

```
if (entry.write_locks) {
  /* soubor je zamčen pro zápis,
   * není možné jej zamknout pro čtení */
} else {
  entry.write_locks++; /* soubor je možné
                        * zamknout pro zápis */
  ...  /* uložení hodnot z entry do databáze */
}
```

Pro zápis. Soubor lze otevřít pro zápis, je-li hodnota čítače zámků pro čtení i zápis nulová. Pokus o zamknutí souboru pro zápis odpovídá následujícímu zjednodušenému pseudokódu:

```
kivfs_entry_t entry;

... /* inicializace proměnné entry pomocí
     * hodnot získaných z databáze */

if ((entry.read_locks + entry.write_locks)) {
  ... /* soubor je zamčen pro čtení nebo zápis,
       * není možné jej zamknout pro zápis */
} else {
  entry.write_locks++; /* soubor je možné
                        * zamknout pro zápis */
  ...  /* uložení hodnot z entry do databáze */
}
```

8.6 Podpora klientské lokální cache

Klientské aplikace na základě četnosti přístupu k souborům vyhodnocují statistiky používání souboru. Získané hodnoty pak používají k rozhodnutí, zda soubor mají vyřadit z lokální cache. Metadatový server četnost přístupů zajišťuje pomocí správy tří čítačů:

Čítač přístupů pro zápis. Jeho hodnota se zvyšuje při:

- vytvoření souboru nebo adresáře,
- otevření souboru v některém z režimů pro zápis,
- změně atributů souboru nebo adresáře.

Čítač přístupů pro čtení. Jeho hodnota se zvyšuje při:

- otevření souboru v režimu pro čtení,
- vylistování obsahu adresáře,
- přečtení atributů souboru nebo adresáře.

Čítač globálních přístupů. Jeho hodnota je rovna součtu hodnot čítačů pro čtení a pro zápis všech souborů.

Hodnoty čítačů přístupů pro čtení a zápis jsou ukládány do tabulky *entries*, ve které se konkrétně se jedná o sloupce *read_hits* a *write_hits*. Jejich hodnoty jsou po výběru z databáze dostupné srkze položky *read_hits* a *write_hits* v datové sturktuře *kivfs_file_t* (příloha C). Klient proto hodnoty obdrží při každém požadavku na:

- výpis obsahu adresáře (požadavek KIVFS_READDIR), kdy server posílá informace o souborech v podobě spojového seznamu, jehož položky jsou právě typu *kivfs_file_t*,
- poskytnutí informací o souboru (požadavek KIVFS_FILE_INFO).

Hodnotu globálního čítače přístupů k souborům metadatový server klientovi poskytuje na vyžádání (v odpovědi na požadavek KIVFS_GLOBAL_HITS).

8.7 Transparentní přístup k šifrovaným datům

Souborový server zabezpečuje a kontroluje uložená data pomocí funkcí pro:

- symetrické šifrování,
- spočítání kontrolních součtů,

dostupných z knihovny *Libgcrypt*, která je poskytována pod licencí GNU/GPL.[16]

8.7.1 Použité algoritmy

K symetrickému blokovému šifrování obsahu souboru souborový server používá algoritmus *Advanced Encryption Standard* s délkou klíče 256 bitů (AES-256). Klíč je uveden v konfiguračním souboru (hodnota parametru *encryption_key*) každého serveru.

Pro vytváření kontrolních součtů obsahu souboru využívá souborový server algoritmus *Secure Hash Algorithm* s délkou výsledného řetězce 512 bitů (SHA-512). Důvodem použití tak složitého algoritmu pro kontrolní součty je snaha o eliminaci případných kolizí, ke kterým dojde při shodném kontrolním součtu dvou různých souborů. Ta je v tomto případě naprosto minimální. V případě nutnosti lze použité algoritmy jednoduše zaměnit za jiné, které jsou knihovnou *Libgcrypt* podporovány.[17]

8.7.2 Transparentní přístup k šifrovaným souborům

Pro zajištění transparentního přístupu k šifrovaným souborům bylo vytvořeno rozhraní, které poskytuje potřebné funkce. Ty jsou uvedeny v příloze B. Pomocí tohoto rozhraní souborový server může:

Otevřít šifrovaný soubor čtení a zápis. Vytváří se dočasný soubor, který obsahuje nezašifrovaný obsah původního souboru. Díky tomu lze v dočasném souboru např. libovolně nastavovat aktuální pozici v souboru a přepisovat data na ní, aniž by bylo nutné se nějak starat o šifrování.

Otevřít šifrovaný soubor zápis. Ze stejného důvodu jako v předchozím bodě se vytváří dočasný soubor o nulové velikosti. Není třeba kopírovat obsah původního souboru, neboť při otevření souboru v režimu jen pro zápis je jeho původní obsah vynulován.

Otevřít šifrovaný soubor čtení. Šifrovaný soubor se otevírá přímo.

Zapisovat do šifrovaného souboru. Zápis je prováděn do nezašifrovaného dočasného souboru.

Číst ze šiforvaného souboru. Čtení je prováděno přímo ze zašifrovaného souboru.

Zavřít šifrovaný soubor. Je-li soubor otevřen režimu pro zápis, pak existuje dočasný soubor, do kterého mohlo být zapisováno. Jestliže se jeho obsah liší od obsahu původního souboru, pak jej musí nahradit. Nahrazení původního souboru probíhá kopírováním dočasného souboru na jeho místo. Během kopírování se zapisovaná data přečtená z dočasného souboru průběžně šifrují. Zároveň se počítá kontrolní součet souboru. Po dokončení nahrazování je dočasný soubor smazán.

Jestliže je soubor otevřen v režimu pro čtení, pak jej stačí zavřít běžným způsobem, neboť se jeho obsah nezměnil a není tedy potřeba přepočítávat kontrolní součet jeho obsahu.

Souborový server kvůli dočasným souborům, se kterými transparentně pracuje, nikdy nezapisuje během přenosu souboru přímo do cílových podsouborů repliky souborů (replik). Značně se tak snižuje pravděpodobnost, že jím spravovaná data budou poškozena vlivem výskytu chyby během přenosu souboru.

8.8 Kvóty virtuálních svazků

V tabulce *volumes* obsahující informace o virtuálních svazcích je ve sloupci *capacity* nastavována hodnota udávající jejich kapacitu. Její hodnotu určuje správce KIVFS, který musí brát na vědomí fakt, že je dána nejmenší kapacitou jednoho z k němu připojených svazků souborových serverů.

Pomocí kapacity virtuálních svazků lze omezit kvóty jednotlivých adresářů z tabulky *entries*, se kterými uživatelé pracují a ukládají do nich soubory. U nově vytvořeného adresáře platí stejná kvóta, která je nastavená u rodičovského adresáře. Správce systému může v databázi definovat, že např. kořenový adresář má kapacitu 10GB a domovské adresáře uživatelů mají kapacitu 30GB. Příklad je uveden v tabulkách č. 7 a 9.

Virtuální svazek	**Kapacita**
Svazek A	10GB
Svazek B	30GB

Tabulka 7: Kapacita virtuálních svazků

Adresář	Připojený virtuální svazek	Výsledná kvóta
/	svazek a	10GB
/home	svazek B	30GB

Tabulka 8: Kvóty adresářů

Aktuálně dostupná kapacita virtuálního svazku, resp. k němu připojených svazků souborových serverů, je dána rozdílem jeho kapacity a celkovým součtem všech souborů, které jsou na něm uložené. Kvůli potřebě rychlého přístupu k těmto informacím byl vytvořen pohled *storage_states* popisující aktuální stavy jednotlivých svazků.

Kvůli relační sémantice sdílení souborů je nutné řešit problém zajištění respektování kvót i v případě, kdy dochází k zápisu do dvou souborů najednou. Protože změny jsou viditelné až po uzavření souboru, je nutné zajistit dodržení kvót. To je zajištěno prostředníctvím rezervace místa na virtuálním svazku a jeho uvažováním při počítání aktuálně dostupné kapacity.

8.8.1 Rezervace místa

Rezervované místo je další vlastnost virtuálních svazků uvedených v tabulce *volumes*. Odpovídá hodnotě ve sloupci *reserved* a udává, kolik bajtů bylo zapsáno během přenosu souboru na server, ale zatím nebylo uloženo.

Před zpracováním každého požadavku od klienta na zápis do repliky souboru souborový server počítá, kolik bajtů je nutno rezervovat. Je-li tato hodnota nenulová, odešle požadavek k rezervaci místa synchronizační vrstvě, která zajistí jeho propagaci na všech metadatových serverech. V případě, že je požadavek úspěšně vyřízen, započítá rezervované místo do celkového počtu rezervovaných bajtů. Pokud je požadavek vyřízen neúspěšně, pak došlo k zaplnění volného místa a informuje o tom klienta.

Souborový server po dokončení zápisu do souboru a jeho zavření odešle celkový počet rezerevovaných bajtů v rámci požadavku na zrušení jejich rezervace (soubor již je nahrán) synchronizační vrstvě. Ta opět zajistí propagaci požadavku na všech metadatových serverech a rezervace místa je zrušena.

8.8.2 Správa kvót

Je-li uživatel začleněn do skupiny admistrátorů, pak má možnost pomocí příkazů (požadavků), které jsou uvedeny v tabulce č. 9, nastavovat kvóty jednotlivých virtuálních svazků.

8.9 Tunelování přenosu souboru

Je-li klient připojen k souborovému serveru a požaduje po něm přenos repliky souboru, kterou nespravuje, pak souborový server musí zprostředkovat transparetní spojení s jiným souborovým serverem, který ji spravuje. Aby mohl vytvořit spojení, v rámci kterého bude zajištěna nejlepší přenosová rychlost, potřebuje znát informace

Příkaz KIVFS	Parametry	Kód	Formát dat	Popis
df		544		Získá informace o kvótách virtuálních svazků.
quota	název virt.svazku kvóta	545	%s %llu	Nastaví kvótu pro virtuální svazek s daným názvem.

Tabulka 9: Příkazy (požadavky) pro správu kvót virtuálních svazků

o aktuálně nejvhodnějším souborovém serveru, ke kterému se má připojit. Ty mu dodává synchronizační vrstva.

Kvůli možnosti výskytu latencí u jednotlivých linek mezi servery, může být místo přímého spojení s cílovým souborovým serverem vhodnější navázat s ním spojení přes další souborový server. Proto informace o aktuálně nejvhodnějším souborovém serveru obsahuje i seznam souborových serverů, přes které je nutné spojení realizovat. Toto spojení je dále označované jako tunel.

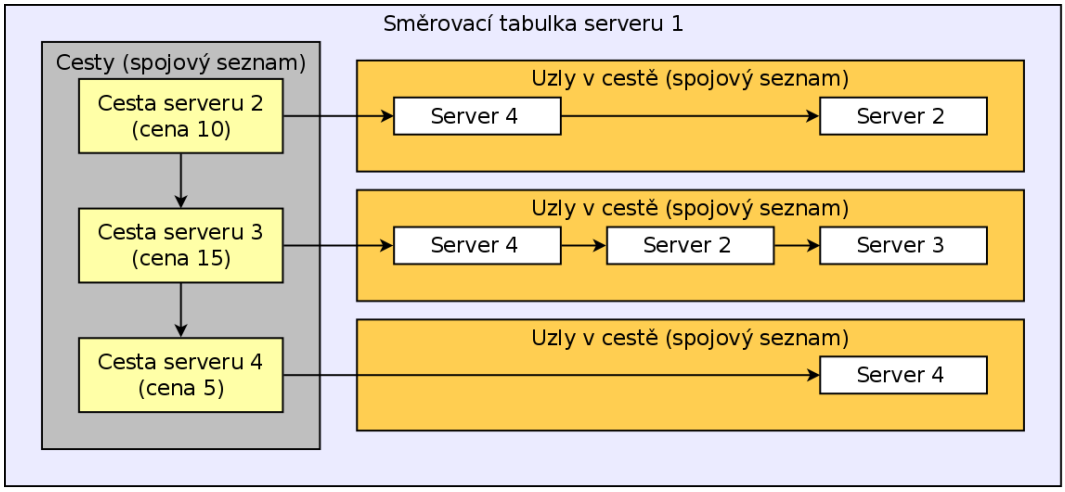
8.9.1 Vytvoření tunelu

Synchronizační vrstva průběžně posílá souborovým serverům informace v podobě směrovací tabulky obsahující informace o aktuálně nejlepších cestách pro realizaci spojení mezi jednotlivými servery (požadavek KIVFS_ROUTE_PUSH). U každé cesty je uvedeno i její ocenění. Směrovací tabulka je realizována spojovým seznamem, jejíž položky jsou níže uvedené datové struktury, a každý souborový server si jí po přijetí uchovává v paměti (obrázek č. 17).

```
typedef struct {
  uint64    id;         // ID uzlu (serveru)
  int       ip;         // IP adresa uzlu
  char      *ip_text;   // IP adresa uzlu v˜textové formě
} kivfs_route_node_t;   // uzel

typedef struct {
  kivfs_route_node_t *dst_node; // cílový uzel
  int32 cost;                   // ocenění cesty
  kivfs_list_t *nodes;          // seznam uzlů v cestě
} kivfs_route_t;                // cesta (položka v routovací tabulce)
```

Klient vyhodnocuje dostupnost všech známých serverů a vždy navazuje spojení k ze svého pohledu nejdostupnějšímu serveru. Přijde-li souborovému serveru, ke kterému je klient právě připojen, požadavek na přenos repliky souboru, kterou nespravuje, vybere na základě porovnání ocenění jednotlivých cest ze své směrovací tabulky nejvhodnější souborový server, jež ji spravuje a také cestu, přes které uzly (souborové servery) je možné se k němu připojit. Poté naváže spojení s prvním uzlem z cesty, kterému odešle požadavek na vytvoření tunelu (KIVFS_CREATE_TUNNEL) obsahující seznam uzlů. Ten z tohoto seznamu odebere sám sebe a pokud není posledním cílovým uzlem, pak jej přepošle dál. Postupně tak dojde k realizaci tunelu.



Obrázek 17: Směrovací tabulka v paměti souborového serveru.

Pokud dojde k situaci, kdy souborový server nenajde žádnou vhodnou cestu, pokusí se realizovat přímé spojení s jedním ze serverů, který repliku spravuje.

Souborový server, ke kterému je klient připojen, přeposílá požadavky na přenos repliky souboru cílovému souborovému serveru přes vytvořený tunel, skrze který následně probíhá i přenos obsahu souboru. Klient má díky tomu zaručenou dostupnost repliky souboru z každého serveru a zároveň nejvyšší možnou rychlost pro přenos jeho obsahu.

8.10 Replikace

Klient zapisuje během přenosu souboru vždy do jeho repliky s aktuální (nejvyšší) verzí. Souborový server spravující tuto repliku souboru po dokončení přenosu zvýší její verzi a skrze synchronizační vrstvu odešle požavek na změnu stavu repliky (požadavek KIVFS_UPDATE_REPLICA) všem metadatovým serverům. Repliky s nižší verzí jsou v tuto chvíli v nekonzistentním stavu (zastaralé). Po úspěšné změně stavu repliky oznámí souborový server ostatním souborovým serverům, že mohou zahájit replikaci (požadavek KIVFS_RUN_REPLICATION). Ostatní souborové servery následně synchronizují všechny jimi spravované zastaralé repliky souboru s jeho aktuální replikou. Postupně tak dojde znovunavrácení všech replik souboru do konzistentního stavu.

Replikaci na každém souborovém serveru zajišťuje na pozadí běžící vlákno, které slouží jako replikační manažer. Proces replikace popsán v následujících podsekcích popisujících jednotlivé kroky, které replikační manažer vykonává.

8.10.1 Zjištění zastaralých replik

Replikační manažer se po přijetí informace, že může zahájit replikaci (požadavek KIVFS_RUN_REPLICATION), dotáže metadatového serveru, které zastaralé

repliky souborů spravuje. Následně zahájí jejich postupnou aktualizaci a obnovu jejich konzistentního stavu.

Aby byla zajištěna replikace i po předcházejícím neúspěšném pokusu o replikaci způsobeném např. nečekaným výpadkem souborového serveru, replikační manažer se průběžně v daných časových intervalech, které jsou definovány v konfiguračním souboru serveru jako hodnota parametru *replication_interval* (výchozí hodnota je 5 minut), dotazuje metadatového serveru, zda nespravuje repliky souboru se zastaralou verzí. Pokud ne, pak jsou všechny repliky, které spravuje, v konzistetním stavu a může čekat, než dostane další oznámení o tom, že může zahájit replikaci (požadavek KIVFS_RUN_REPLICATION), nebo než vyprší další časový interval pro replikaci. V opačném případě musí zahájit jejich již zmíněnou postupnou aktualizaci a obnovu konzistentního stavu.

8.10.2 Zjištění aktuálních replik

Po vyhodnocení, že spravuje zastaralou repliku souboru, která je v nekonzistentním stavu, musí zjistit, z jakých zdrojů lze stáhnout její aktuální verzi. Dotazuje se proto metadatového serveru na seznam aktuálních replik souboru včetně informací, které souborové servery je spravují.

Následně vybere první souborový server, který aktuální repliku souboru spravuje a požádá skrze synchronizační vrstvu o zamčení souboru pro čtení. Poté opět od metadatového serveru zjistí informace o podsouborech své zastaralé repliky souboru a také o podsouborech aktuální repliky souboru spravované jiným serverem.

Díky tomu, že informace o podsouboru repliky obsahuje i kontrolní součet jeho obsahu, může zastaralé a aktuální podsoubory dle nich mezi sebou porovnat. Tím získá seznam podsouborů, které byly v aktuální replice souborou změněny a je potřeba je v zastaralé replice souboru aktualizovat. Replikační manažer tak efektivně šetří datové přenosy.

8.10.3 Aktualizace zastaralé repliky souboru

Replikační manažer spravující zastaralou repliku souboru naváže spojení s vybraným souborovým serverem, který spravuje aktuální repliku souboru, a pošle mu požadavek (KIVFS_PROVIDE_REPLICA) na poskytnutí obsahu seznamu požadovaných podsouborů.

Po jejich stáhnutí a uložení mají podsoubory zastaralé repliky souboru stejný obsah jako podsoubory aktuální repliky souboru. Nastavením aktuální verze zastaralé replice souboru se z ní stává další aktuální replika souboru.

8.10.4 Dokončení aktualizace zastaralé repliky souboru

Podobně jako po dokončeném přenosu souboru z klienta na souborový server, replikační manažer odešle skrze synchronizační vrstvu informace o jeho aktualizaci a požadavek na odemčení souboru pro čtení.

V tuto chvíli replikační manažer úspěšně dokončil replikaci daného souboru. Pokud spravuje další zastaralé repliky, celý proces zopakuje. V opačném případě čeká na další vypršení časového intervalu k replikaci.

8.11 Deduplikace

Experimentálně implementovanou funkcí je deduplikace uložených podsouborů. Ta je na každém souborovém serveru realizována v rámci deduplikačního vlákna, které průběžně ověřuje, zda nemají stejný obsah nějaké podsoubory uložené na stejném svazku souborového serveru.

Deduplikovaný podsoubor je podsoubor, ke kterému odpovídá záznam v tabulce *sources*. Deduplikační vlákno se pokouší o deduplikaci podsouborů na jednotlivých svazcích souborového serveru v daných časových intervalech, které jsou definovány v konfiguračním souboru serveru jako hodnota parametru *deduplication_interval* a jehož výchozí hodnota je 90 minut. Důvodem pro použití velkého intervalu je časová a procesorová náročnost deduplikace podsouborů, která není vyžadována jako nutná funkčnost systému a v současné verzi KIVFS jde o experimentální funkci[9]. Průběh deduplikace podsouborů je popsán v následujících podsekcích.

8.11.1 Zjištění zdrojových kandidátů

V relační databázi spravované metadatovým serverem je uložený pohled *source_candidates*, s jehož pomocí jsou získávány informace o kandidátech použitelných jako zdrojové podsoubory. Ty jsou vybíráni z podsouborů z tabulky *parts* umístěných na stejných svazcích souborového serveru (tabulka *storages*) a mají následující specifické vlastnosti:

- shodný kontrolní součet obsahu s jiným podsouborem, který ale nesmí být shodný s kontrolním součtem již existujícího zdrojového podsouboru (pokud by se kontrolní součty shodovaly, jednalo by se o další nalezený duplicitní podsoubor),
- musí mít nenulovou velikost.

Deduplikační vlákno se v prvním kroku dotazuje metadatového serveru, zda nezjistil nějaké podsoubory, kteřé mohou sloužit jako deduplikační kandidáty. Pokud ano, obdrží jejich seznam a pro každý z nich provede následující operace:

1. požádá metadatový server o zamčení odpovídajícího souboru pro čtení,
2. v adresáři, ve kterém se podsoubor (deduplikační kandidát) nachází, vytvoří adresář *sources*,
3. v adresáři *sources* vytvoří kopii podsouboru s názvem, který odpovídá jeho kontrolnímu součtu vyhodnocenému pomocí SHA-512,
4. odešle informace o vytvořeném zdrojovém podsouboru metadatovému serveru, který vytvoří nový záznam v tabulce *source_parts*,
5. požádá metadatový server o odemčení odpovídajícího souboru pro čtení.

[9] Ideální naplánování spuštění deduplikace podsouborů by se mohlo odvíjet např. od aktuálního zatížení systému.

8.11.2 Deduplikace podsouborů

Dalším pohledem v relační databázi spravované metadatovým serverem je *duplicated_parts*. pomocí kterého lze získat seznam duplicitních podsouborů z tabulky *parts*. Duplicitní podsoubory jsou identifikovány dle shody jejich kontrolního součtu s kontrolním součtem zdrojového podsouboru z tabulky *sources*.

V druhém kroce se deduplikační vlákno dotazuje metadatového serveru, zda existují duplicitní podsoubory. Pokud ano, obdrží jejich seznam a pro každý z nich provede:

1. požádá metadatový server o uzamčení odpovídajícího souboru pro zápis,
2. zjistí zdrojový podsoubor se stejným kontrolním součtem,
3. deduplikuje obsah podsouboru (vynuluje jej),
4. odešle informace o nově deduplikovaném souboru obsahující informace o zdrojovém souboru,
5. požádá metadatový server o odemčení odpovídajícího souboru pro zápis.

8.11.3 Odstranění nepoužitých zdrojových souborů

V posledním kroku se deduplikační vlákno dotazuje metadatového serveru, zda neexistuje nějaký nepoužitý zdrojový soubor, který je možné smazat. Metadatový server provedením dotazu do databáze zjistí všechny zdrojové podsoubory, na které se neodkazují žádné deduplikované podsoubory a odešle jejich seznam zpět deduplikačnímu vláknu. To pak postupně smaže všehnny zbytečné zdrojové podsoubory a informuje o tom metadatový server, který je také smaže a uvolní tak místo pro další soubory uživatelů.

8.11.4 Otevření deduplikovaného podsouboru

Souborový server před otevřením podsouboru pozná dle příznaku[10], zda se jedná o deduplikovaný podsoubor či nikoliv. Pokud se jedná o deduplikovaný podsoubor, pak musí s pomocí metadatového serveru zjistit, kterému zdrojovému podsouboru odpovídá a v případě, že jej chce otevřít:

- pro čtení, musí nahradí cestu k deduplikovanému souboru cestou ke zdrojovému souboru,
- pro zápis, musí zkopírovat obsah zdrojového souboru do deduplikovaného souboru (tzn. duplikovat jeho obsah).

Teprvé poté může deduplikovaný podsoubor otevřít a pracovat s ním jako s běžným podsouborem.

[10]Nenulové číslo odpovídající jednoznačnému identifikátoru zdrojového souboru.

8.12 Implementace logování požadavků

Metadatový server podporuje logování požadavků na požádání a jejich obsah ukládá do tabulky *message_logs*. Prostřednictvím poskytování následujících operací:

- zalogování zprávy s příslušnou časovou známkou (logický čas),
- zjištění zalogovaných zpráv, jejichž časová známka patří do daného časového intervalu,
- smazání zalogovaných zpráv, jejichž časová známka patří do daného časového intervalu,

zprostředkovává správu všech zalogovaných požadavků.

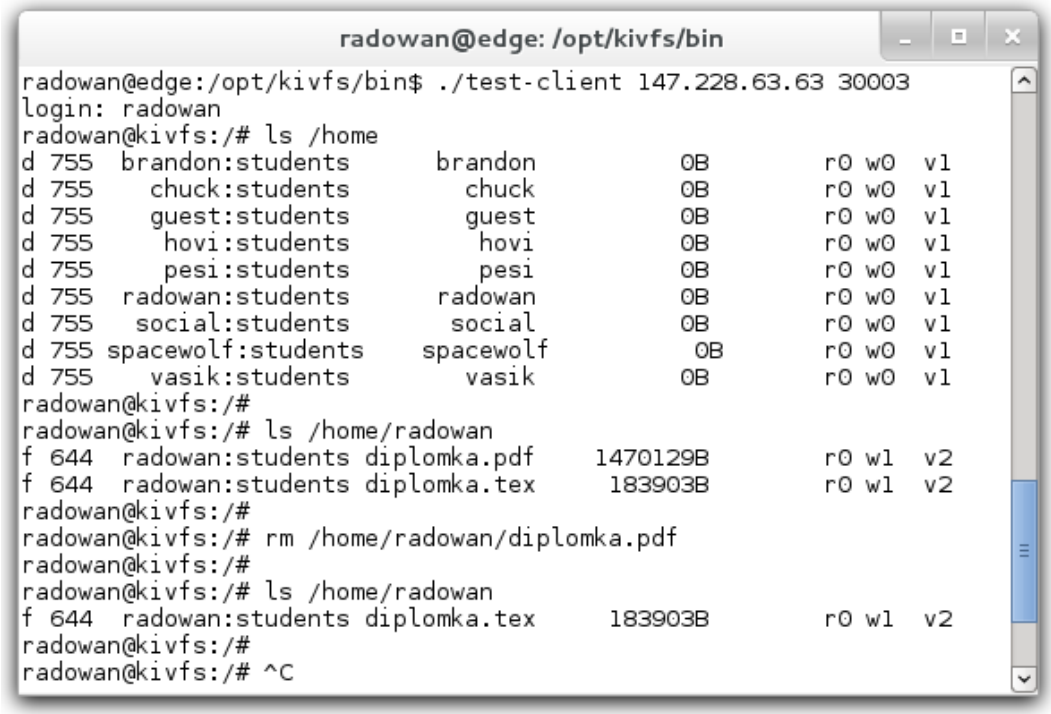
Implementace funkce logování požadavků je stěžejní funkce pro synchronizační vrstvu, která potřebuje na všech serverech shodně logovat požadavky na změnu uložených dat kvůli případné obnově po výpadku serveru.[20]

9 Implementace vlastního klienta

Během vývoje vznikla potřeba okamžitě testovat čerstvě implementované služby poskytované datovým úložištěm. Z toho důvodu byl implementován konzolový klient.

Klient je naprogramován v jazyce C a plně využívá funkce dostupné z knihovny *libkivfscore* (sekce 7). Díky tomu zároveň slouží i jako referenční klient, ze kterého vycházejí všechny ostaní implementace klientských aplikací.

Pro uživatele klient poskytuje rozhraní podobné unixovému shellu (obrázek č. 18). V příloze E jsou popsány všechny funkce datového úložiště, které lze pomocí jednotlivě zadávaných příkazů testovat.



Obrázek 18: Screenshot testovacího klienta.

10 Výkonnostní testy

Výkonnostní testy datového úložiště byly provedeny na sestavách uvedených v tabulkách č. 10 a 11, které jsou vzájemně propojené v rámci lokální sítě, která podporuje přenosovou rychlost až 100Mbps. Všechny měření byly 5-krát opakovány. Výsledné hodnoty byly odvozeny z aritmetického průměru naměřených hodnot.

1. Server	
Procesor	2x AMD Athlon(tm) 64 X2 Dual Core Processor 5200+
Paměť	4063MB
Operační systém	Debian GNU/Linux wheezy/sid (Linux 3.1.0-1-amd64, x86_64)
Pevný disk	ATA ST3500418AS (kapacita 465.76 GiB)
Souborový systém	EXT4

Tabulka 10: Testovací sestava (server č. 1).

2. Server a klient	
Procesor	AMD Sempron(tm) 140 Processor
Paměť	1021MB
Operační systém	Ubuntu 11.10 (Linux 3.0.0-12-generic, x86_64)
Pevný disk	ATA ST3160812A (kapacita 465.76 GiB)
Souborový systém	EXT4

Tabulka 11: Testovací sestava (server č. 2 a klient).

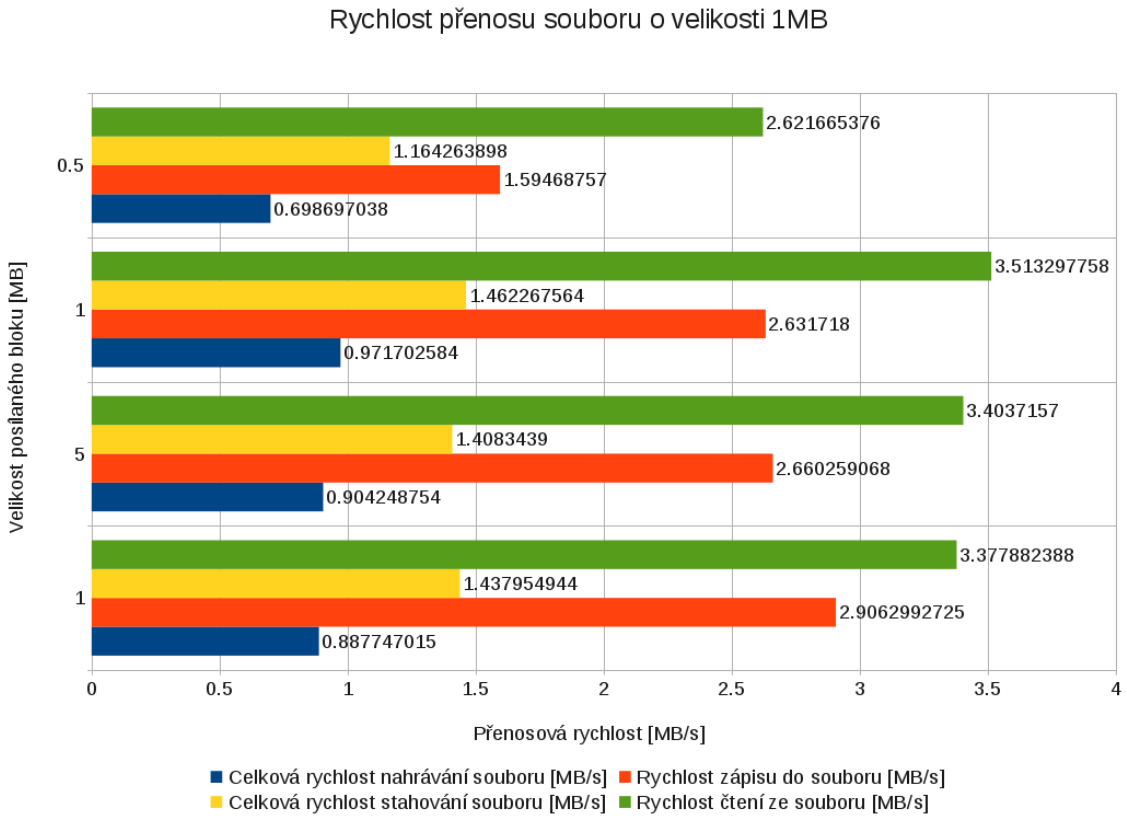
10.1 Přenosová rychlost

V následujících tabulkách č. 12 až 15 a grafech na obrázcích č. 19 až 22 je uvedena naměřená přenosová rychlost pro upload a download souborů o velikosti 1MB, 10MB, 100MB a 1000MB, které byly nahrávány po blocích o velikosti 512KB, 1MB, 5MB a hodnotě odpovídající velikosti souboru. Maximální velikost podsouboru na souborovém serveru je 50MB. Obsah uložených podsouborů na straně serverů byl šifrován a byl počítán jeho kontrolní součet.

V testech je uvažováno, že nedochází ke zpoždění během synchronizace požadavků (činnost synchronizační vrstvy). Klient se proto během testování připojuje přímo k VFS serveru, který pracuje v lokálním režimu a nepřeposílá požadavky synchronizační vrstvě.

Velikost posílaného bloku souboru [MB]	Celková rychlost nahrávání souboru [MB/s]	Rychlost zápisu do souboru [MB/s]	Celková rychlost stahování souboru [MB/s]	Rychlost čtení ze souboru [MB/s]
1	0.887747015	2.9062992725	1.437954944	3.377882388
5	0.904248754	2.660259068	1.4083439	3.4037157
1	0.971702584	2.631718	1.462267564	3.513297758
0.5	0.698697038	1.59468757	1.164263898	2.621665376

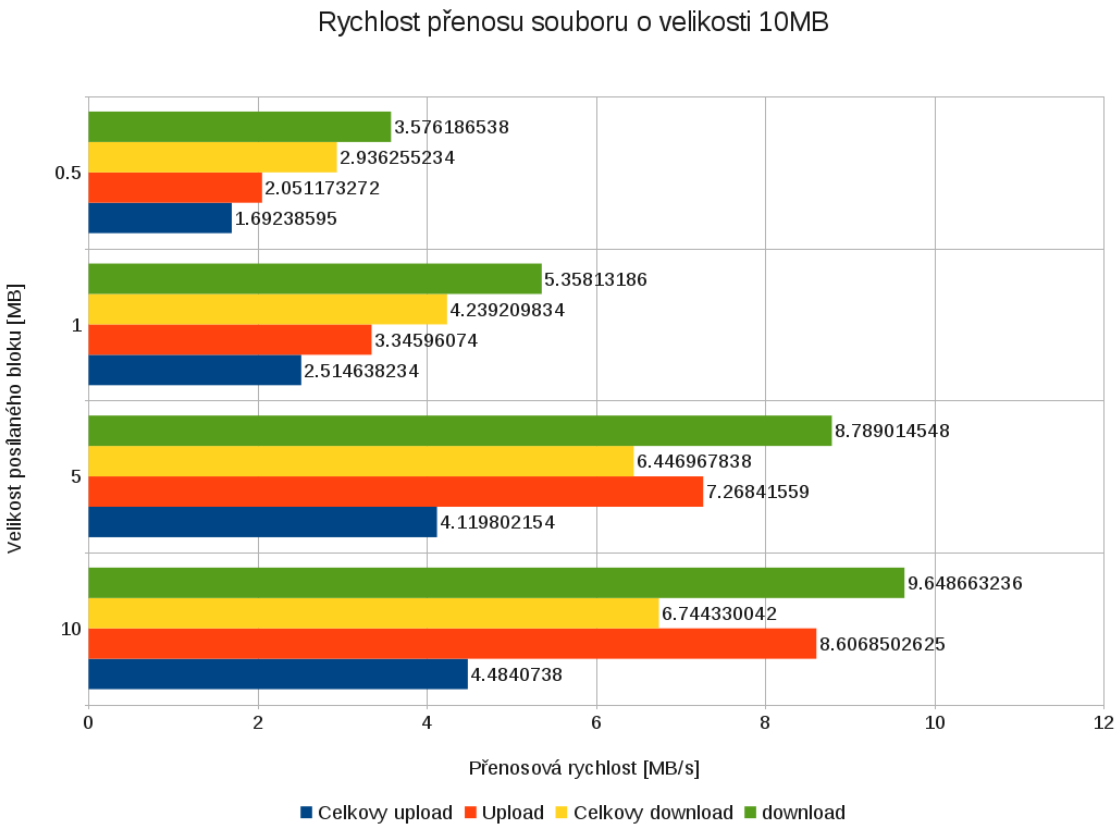
Tabulka 12: Výsledné hodnoty z testu rychlosti přenosu 1MB souboru.



Obrázek 19: Graf výsledných hodnot z testu rychlosti přenosu 1MB souboru.

Velikost posílaného bloku souboru [MB]	Celková rychlost nahrávání souboru [MB/s]	Rychlost zápisu do souboru [MB/s]	Celková rychlost stahování souboru [MB/s]	Rychlost čtení ze souboru [MB/s]
10	4.4840738	8.6068502625	6.744330042	9.648663236
5	4.119802154	7.26841559	6.446967838	8.789014548
1	2.514638234	3.34596074	4.239209834	5.35813186
0.5	1.69238595	2.051173272	2.936255234	3.576186538

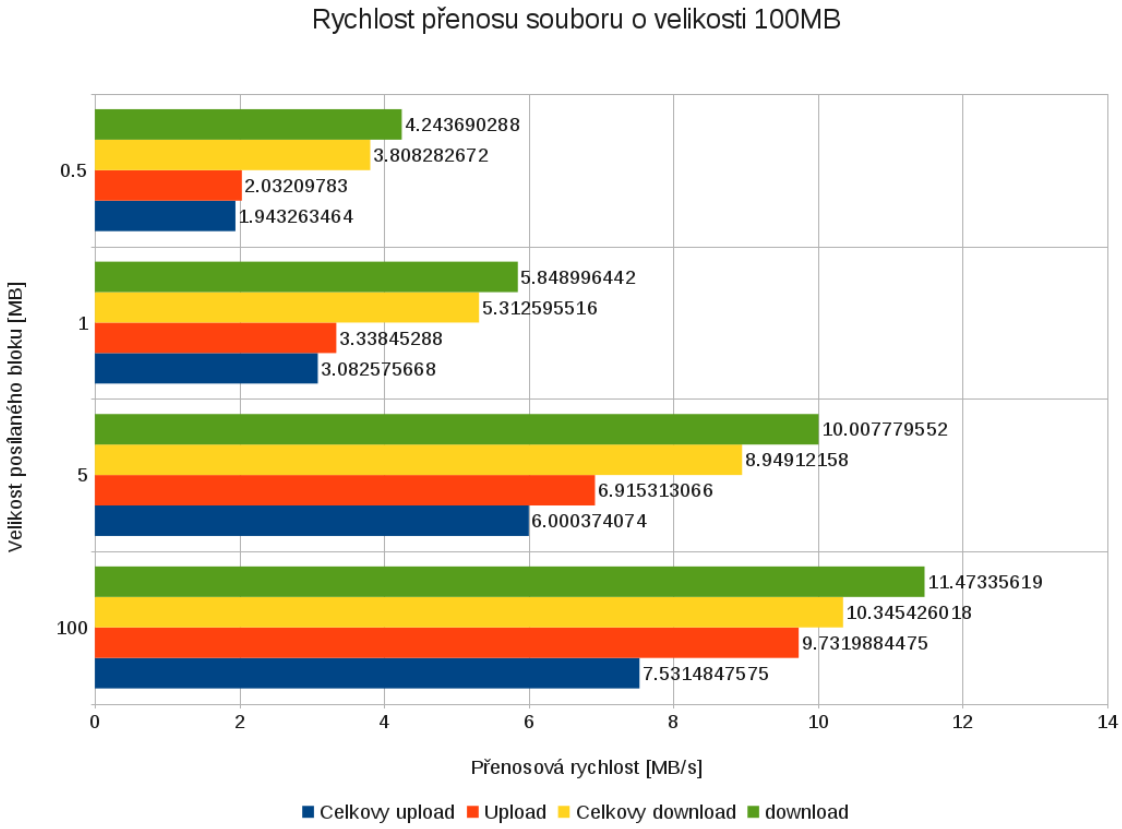
Tabulka 13: Výsledné hodnoty z testu rychlosti přenosu 10MB souboru.



Obrázek 20: Graf výsledných hodnot z testu rychlosti přenosu 10MB souboru.

Velikost posílaného bloku souboru [MB]	Celková rychlost nahrávání souboru [MB/s]	Rychlost zápisu do souboru [MB/s]	Celková rychlost stahování souboru [MB/s]	Rychlost čtení ze souboru [MB/s]
100	7.5314847575	9.7319884475	10.345426018	11.47335619
5	6.000374074	6.915313066	8.94912158	10.007779552
1	3.082575668	3.33845288	5.312595516	5.848996442
0.5	1.943263464	2.03209783	3.808282672	4.243690288

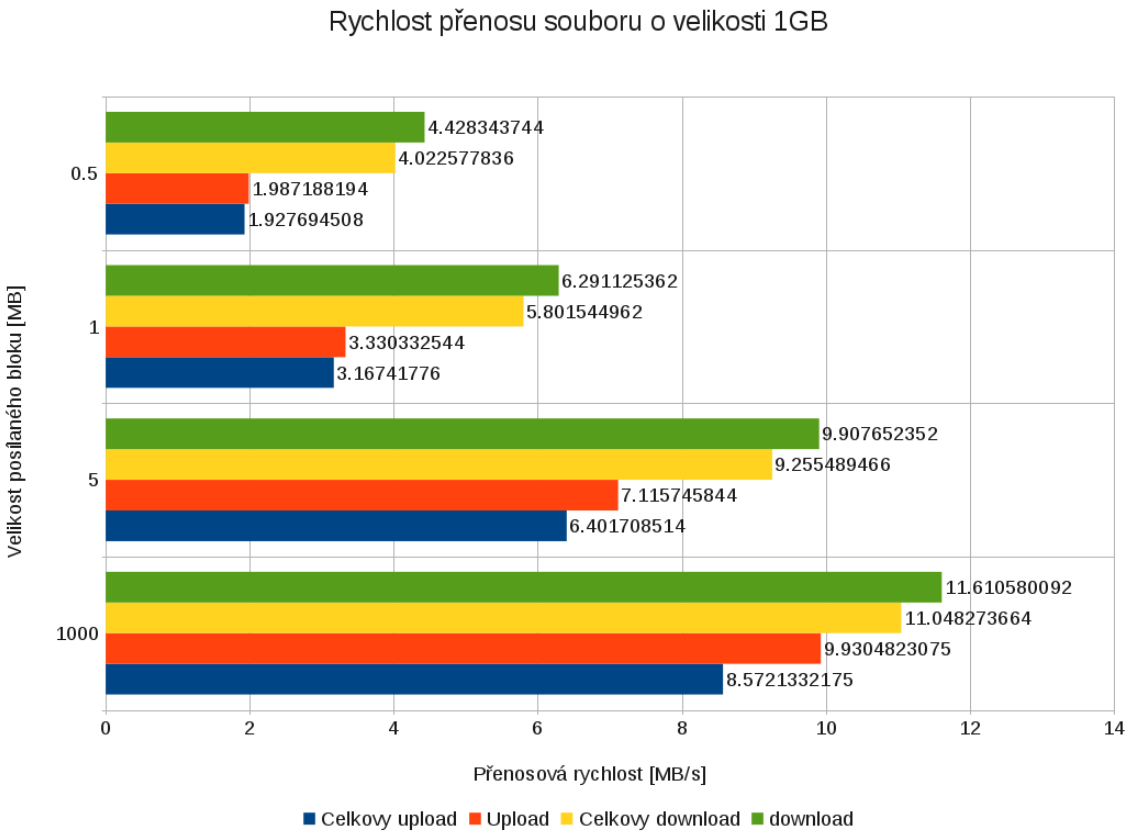
Tabulka 14: Výsledné hodnoty z testu přenosové rychlosti 100MB souboru.



Obrázek 21: Graf výsledných hodnot z testu rychlosti přenosu 100MB souboru.

Velikost posílaného bloku souboru [MB]	Celková rychlost nahrávání souboru [MB/s]	Rychlost zápisu do souboru [MB/s]	Celková rychlost stahování souboru [MB/s]	Rychlost čtení ze souboru [MB/s]
1000	8.5721332175	9.9304823075	11.048273664	11.610580092
5	6.401708514	7.115745844	9.255489466	9.907652352
1	3.16741776	3.330332544	5.801544962	6.291125362
0.5	1.927694508	1.987188194	4.022577836	4.428343744

Tabulka 15: Výsledné hodnoty z testu rychlosti přenosu 1GB souboru.



Obrázek 22: Graf výsledných hodnot z testu rychlosti přenosu 1GB souboru.

10.1.1 Vyhodnocení výsledků

Z grafů je patrné, že přenos je ovlivněn režií během otevírání a zavírání souboru a také velikostí bloků, po kterých klient nahrává nebo stahuje soubor. Vzhledem k tomu, že klient ví, kolik bajtů ze souboru chce přesně přenést[11] a tedy i jakou má

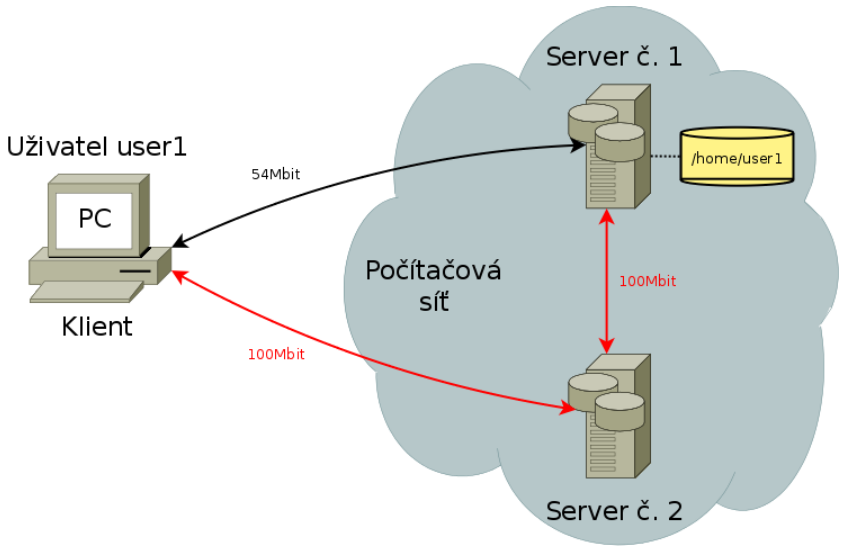
[11] Pracuje vždy s konkrétní částí souboru, která může tvořit i celý jeho obsah.

posílat velikost bloku, lze konstatovat, že při operaci zápis či čtení dosáhne vždy nejvyšší možné přenosové rychlosti a oproti původní verzi KIVFS je jeho přístup k souborům efektivnější díky možnosti přistupovat k pouze požadované části obsahu souboru.

10.2 Tunelované spojení

Na obrázku č. 23 je znázorněné schéma lokální sítě, v rámci které bylo testováno tunelované spojení. Klient má přímé spojení ke dvěma serverům. Jeho data spravuje server, ke kterému má přístup skrze 54Mbit linku (bezdrátová síť). Ke druhému serveru, který nastaven tak, že právě nespravuje žádná data uživatelů, má klient přístup skrze 100Mbit linku.

Je-li klient připojen k prvnímu serveru, pak transparentně přistupuje k datům skrze přímé spojení. Připojí-li se klient druhému serveru, pak je jeho transparentní přístup k datům tunelován skrze spojení mezi servery.

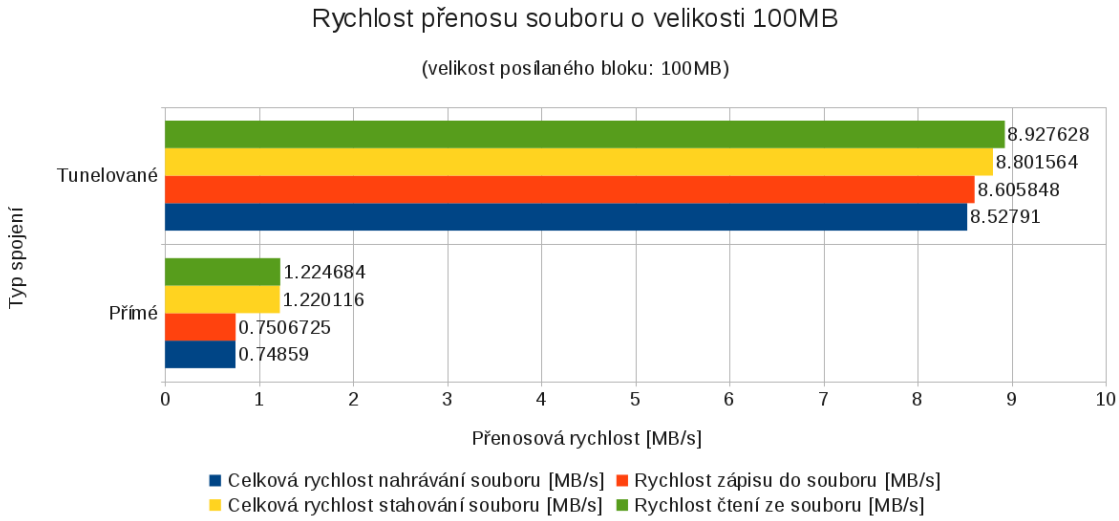


Obrázek 23: Schéma tunelování spojení (100Mbit tunel vyznačen červeně).

V následující tabulce č. 16 a grafu č. 23 jsou uvedeny hodnoty, které byly získány během měření při přenosu souborů skrze přímé spojení a následně skrze tunelované spojení mezi klientem a cílovým serverem. Klient přenáší soubor o velikosti 100MB po blocích odpovídajících hodnotě velikosti souboru. Maximální velikost podsouboru na souborovém serveru je 50MB. Obsah uložených podsouborů na straně serverů byl šifrován a byl počítán jeho kontrolní součet.

Typ spojení	Celková rychlost nahrávání souboru [MB/s]	Rychlost zápisu do souboru [MB/s]	Celková rychlost stahování souboru [MB/s]	Rychlost čtení ze souboru [MB/s]
Tunelované	0.74859	0.7506725	1.220116	1.224684
Přímé	8.52791	8.605848	8.801564	8.927628

Tabulka 16: Výsledné hodnoty z testu rychlosti přenosu 100MB souboru srkze přímé a tunelované spojení.



Obrázek 24: Graf výsledných hodnot z testu rychlosti přenosu 100MB souboru srkze přímé a tunelované spojení.

10.2.1 Vyhodnocení výsledků

Z grafu je patrné, že rychlost přenosu souboru skrze tunelované spojení je značně vyšší. Rychlost přenosu skrze přímé spojení odpovídá nejvyšší možné přenosové rychlosti, kterého bylo možné dosáhnout před implementací mechanismu tunelování spojení. Lze konstatovat, že celková propustnost systému je oproti původní verzi KIVFS mnohem větší.

11 Organizace zdrojových kódů a jejich překlad

Zdrojové kódy knihovny libkivfscore a datového úložiště KIVFS jsou organizovány v samostatných adresářích a distribuovány společně v archivu *KIVFS-VFS.tar.gz*, který je k dispozici na přiloženém CD. Jeho obsah lze extrahovat pomocí příkazu:

```
tar -xvvzf KIVFS-VFS.tar.gz
```

Po jehož vykonání v aktuálním pracovním adresáři přibyly:

- adresáře obsahující zdrojové kódy VFS (*./vfs/*), metadatového (*./db/*) a souborového serveru (*./fs/*),
- adresář obsahující zdrojové kódy knihovny libkivfscore (*./core/*),
- adresář obsahující zdrojové kódy testovacího klienta (*./client/*),
- soubor obsahující vzorový SQL dump databáze (*./kivfs_db.sql*),
- vzorový konfigurační soubor(*./kivfs.conf*),
- instalační balíčky (*kivfs-vfs.deb* a *kivfs-core.deb*),
- instalační skript(*./install.sh*).

Každý adresář se zdrojovými kódy serveru obsahuje:

- zdrojové soubory,
- adresář *include*, který obsahuje hlavičkové soubory,
- soubor Makefile, pomocí kterého lze přeložit zdrojové kódy.

Příklad zdrojových kódů vfs serveru:

```
vfs
|-- include
|   |-- kivfs-vfs-forward.h
|   `-- kivfs-vfs-thread.h
|-- kivfs-vfs-forward.c
|-- kivfs-vfs-server.c
|-- kivfs-vfs-thread.c
`-- Makefile
```

11.1 Překlad a instalace

Pro úspěšný překlad a následné spuštění je nutné mít v systému nainstalován následující software:

- překladač *gcc*,
- program *make*,
- databázový server *MySQL*,

- knihovnu *libgcrypt*,
- knihovnu *libmysqlclient*,
- knihovnu *libssl*,

které lze v rámci operačního systému GNU/Linux (distribuce Debian) nainstalovat prostřednictvím následujícího příkazu spuštěného:

```
apt-get install gcc make mysql-server libgcrypt11-dev libmysqlcliet-dev \
  libssl-dev
```

Datové úložiště KIVFS lze nainstalovat z balíčků prostřednictvím spuštění příkazu:

```
dpkg -i kivfs-core.deb kivfs-vfs.deb
```

nebo prostřednictvím instalačního skriptu, který zajistí překlad zdrojových souborů na spustitelné soubory a zajistí jejich instalaci do systému. Instalační skript lze spusit pomocí příkazu:

```
sh ./install.sh
```

Po úspěšné instalaci, během které byla automaticky vytvořena i databáze KIVFS:

- knihovna libkivfscore je nainstalována v systému (v adresáři */usr/lib*),
- spustitelné soubory jednotlivých serverů jsou umístěné v adresáři */opt/kivfs/bin*,
- vzorový konfigurační soubor je dostupný v adresáři */opt/kivfs/etc*.

11.2 Spuštění

Jednotlivé servery lze spustit nezávisle na vyšších vrstvách KIVFS pomocí následujících příkazů:

```
/opt/kivfs/bin/kivfs-vfs-server /opt/kivfs/etc/kivfs.conf
/opt/kivfs/bin/kivfs-db-server /opt/kivfs/etc/kivfs.conf
/opt/kivfs/bin/kivfs-fs-server /opt/kivfs/etc/kivfs.conf
```

Příklad konfiguračního souboru je uveden v přiloze F. Testovacího klienta lze spustit pomocí následujícího příkazu:

```
/opt/kivfs/bin/test-client <IP adresa VFS serveru> <port VFS serveru>
```

Servery průběžně informují o své činnosti prostřednictvím ukládání zpráv do systémového logu.

12 Závěr

Cílem práce bylo navrhnout a implementovat řešení vedoucí k odstranění slabých stránek KIVFS z hlediska požadavků na rozsáhlá datová úložiště, které bránily dalšímu rozšíření a byly úzkým hrdlem systému.

Z hlediska datového úložiště KIVFS a jím poskytovaných služeb byla zvýšena podpora mobilních platforem, pro které bylo přizpůsobeno přistupování k souborům (práce se souborem po částech), přidána podpora ACL a lokální cache ze strany serveru pomocí uchovávání hodnot pro statisku přístupů k souborům, atd. Na straně serveru byla zvýšena bezpečnost uložených dat uživatelů prostřednictvím šifrování obsahu svazků souborových serverů, přibyla možnost přidělovat kvóty k jednotlivým virtuálním svazkům a k nim připojeným adresářům umožňující omezit uživatele. Dále pak přibyla možnost rozdělovat soubory do menších částí, během přenosu souborů tunelovat spojení mezi souborovými servery, logovat zprávy pro případnou obnovu serveru po jeho výpadku. Služby poskytované datovým úložištěm byly rozšířeny o možnost např. zjistit rozšířené atributy souboru (práva, statistika, atd.), upravit nastavení kvót, apod.

Současná verze datového ůložiště je optimalizována z hlediska multimaster online replikace, správy uložených dat v databázi a navíc byla přidána experimentální funkce pro deduplikaci uložených souborů, která se zatím u žádného jiného běžně používaného DFS nevyskytuje. Průběžné zátěžové testy ověřily stabilitu úložiště a výsledky výkonnostních testů prokázály jeho výkonnost i při šifrování obsahu souborových svazků serverů.

Práce splňuje všechny původně kladené požadavky a nad její rámec byl implementován testovací klient KIVFS, který nyní slouží i jako referenční aplikace pro další vývoj ostatních klientů či nástrojů pro správu systému. Dále byla vytvořena knihovna libkivfscore, která usnadňuje a urychluje rozvoj celého KIVFS.

Z hlediska dalšího vývoje projektu KIVFS lze pokračovat v souvislosti s datovým úložištěm v jeho optimalizaci a dalším rozšiřování poskytovaných funkcí. Ty se mohou týkat např. komprese přenášených dat a to jak mezi klientem a serverem, tak i servery samotnými, plánování spouštění deduplikace, které bude závislé na vyhodnocení aktuální zátěže systému, atd. Jejich realizace může být provedena v rámci dalších semestrálních, bakálářských a diplomových prací.

13 Přehled zkratek a použitého značení

Přehled zkratek a použitého značení

Zkratka	Celý název
FS	File system
VFS	Virtual file system
DFS	Distributed file system
ACL	Access control list
GPL	General public license
BSD	Berkeley software distribution
ERA	Entity relationship attribute diagram
SQL	Structured query language
TCP	Transmission control protocol
NAT	Network address translation
IP	Internet protocol
LVM	Logical volume management (LVM)
RAID	Redundant array of inexpensive disks
DDoS	Distributed denial of service
FUSE	Filesystem in userspace

Tabulka 17: Přehled zkratek a použitého značení.

Literatura

[1] *OpenAFS:* **OpenAFS Administration Guide** [online]
URL: <http://docs.openafs.org/AdminGuide>
[cit. 2012-05-17]

[2] *BRAAM, Peter J.:* **The Coda Distributed File System** [online]
URL: <http://www.coda.cs.cmu.edu/ljpaper/lj.html>
[cit. 2012-05-17]

[3] *Center for Information Technology Integration:* **NFS Version 4 Open Source Reference Implementation** [online]
URL: <http://www.citi.umich.edu/projects/nfsv4>
[cit. 2012-05-17]

[4] *Network Working Group:* **Internet Small Computer Systems Interface (iSCSI)** [online]
URL: <http://www.ietf.org/rfc/rfc3720.txt>
[cit. 2012-05-17]

[5] *Hellman, Brian, HAAS, Florian, REISNER, Philipp, ELLENBERG, Lars:* **The DRBD User's Guide** [online]
URL: <http://www.drbd.org/users-guide/>
[cit. 2012-05-17]

[6] *MEGGYESI, Zoltán:* **Fibre Channel Overview** [online]
URL: <http://hsi.web.cern.ch/HSI/fcs/spec/overview.htm>
[cit. 2012-05-17]

[7] *TANENBAUM, Andrew S.:* **Distributed systems: principles and paradigms. 2nd ed.**
Upper Saddle River: Prentice Hall, 2006. 686 s. ISBN 0-13-239227-5.

[8] *COULOURIS, George, DOLLIMORE, Jean a KINDBERG, Tim:* **Distributed systems: concepts and design. 3rd ed.**
Harlow: Addison-Wesley, 2001. xiii, 772 s. ISBN 0-201-61918-0

[9] *CHOW, Randy a JOHNSON, Theodore:* **Distributed operating systems and algorithms. [1st ed.]**
Reading: Addison-Wesley, 1998. xx, 569 s. ISBN 0-201-49838-3

[10] *MULLENDER, Sape:* **Distributed systems. 2nd ed.**
Addison-Wesley: Wokingham, 1995. xvi, 601 s. ISBN 0-201-62427-3

[11] *CAMPBELL, Richard:* **Managing AFS: the Andrew File System [1st ed.]**
New Jersey: Prentice-Hall, 1998. xvi, 479 s. ISBN 0-13-802729-3

[12] *ŠVAMBERG, Michal:* **OpenAFS** [online]
URL: <http://abclinuxu.cz/serialy/openafs>
[cit. 2012-05-17]

[13] *JUNÁK, M., MATĚJKA, L., PEŠIČKA, L., PIVNIČKA, M., SKUPA, J., STREJC, R., STEINER, V., ŠAFAŘÍK, J.:* **KIV-DFS-Experimental Distributed File System**
In Informatics 2009. Košice: Technical University, 2009. s. 45-50. ISBN: 978-80-8086-126-1

[14] *Network Working Group:* **The MD5 Message-Digest Algorithm** [online]
URL: <http://tools.ietf.org/html/rfc1321>
[cit. 2012-05-17]

[15] *Network Working Group:* **US Secure Hash Algorithms (SHA and HMAC-SHA)** [online]
URL: <http://tools.ietf.org/html/rfc1321>
[cit. 2012-05-17]

[16] *GnuPG:* **Libgcrypt** [online]
URL: <http://www.gnupg.org/related_software/libraries.en.html>
[cit. 2012-05-17]

[17] *Free Software Foundation, Inc.:* **The Libgcrypt Reference Manual** [online]
URL: <http://www.gnupg.org/documentation/manuals/gcrypt/>
[cit. 2012-05-17]

[18] *STREJC, Radek:* **KIVFS - Souborový a databázový server**
Plzeň, 2009. Bakalářská práce. Západočeská univerzita, Fakulta aplikovaných věd. Vedoucí práce Luboš Matějka.

[19] *PIVNIČKA, Marek:* **KIVFS - Zabezpečení, šifrování a ověření identity**
Plzeň, 2009. Bakalářská práce. Západočeská univerzita, Fakulta aplikovaných věd. Vedoucí práce Luboš Matějka.

[20] *SKUPA, Jindřich:* **KIVFS - Synchronizace a trasování požadavků**
Plzeň, 2012. Diplomová práce. Západočeská univerzita, Fakulta aplikovaných věd. Vedoucí práce Luboš Matějka.

[21] *FAZEKAŠ, Jan:* **KIVFS - Klient pro GNU/Linux s pomocí jaderného modulu**
Plzeň, 2010. Bakalářská práce. Západočeská univerzita, Fakulta aplikovaných věd. Vedoucí práce Luboš Matějka.

[22] *JAROŠ, Přemysl:* **KIVFS - Klient pro GNU/LINUX s použitím FUSE**
Plzeň, 2012. Diplomová práce. Západočeská univerzita, Fakulta aplikovaných věd. Vedoucí práce Luboš Matějka.

[23] *STEINER, Václav:* **KIVFS - File Manager pro MS Windows**
Plzeň, 2012. Bakalářská práce. Západočeská univerzita, Fakulta aplikovaných věd. Vedoucí práce Luboš Matějka.

[24] *Portable Applications Standards Committee of the IEEE Computer Society:* **Draft Standard for Information Technology - Portable Operating System Interface (POSIX) - Part 1: System Application Program Interface (API) - Amendment #: Protection, Audit and Control Interfaces [C Language] IEEE 1003.1e. Draft 17** [online]
URL: <www.suse.de/ agruen/acl/posix/Posix_1003.1e-990310.pdf>
[cit. 2012-05-17]

14 Přílohy

Příloha A - Funkce pro správu souboru v databázi

```
/* vybere adresář v adresáři s daným názvem */
int db_get_directory(db_transaction_t *transaction, // id. transakce
                     kivfs_entry_t **p_entry,       // vybraný adresář
                     kivfs_entry_t *parent_entry,   // rodičovský adresář
                     char *name);                   // název adresáře

/* vybere soubor/adresář s daným ID */
int db_get_file_with_id(db_transaction_t *transaction, // id. transakce
                        kivfs_entry_t **p_entry,       // vybraný soubor
                        uint64 entry_id);              // ID souboru

/* vybere seznam souborů obsažených v adresáři */
int db_get_list_directory(db_transaction_t *transaction, // id. transakce
                          kivfs_list_t **p_files,        // vybraný seznam
                                                         // souborů
                          kivfs_entry_t *parent_entry);  // rodičovský adresář

/* smaže soubor (adresář) s daným ID */
int db_delete_entry(db_transaction_t *transaction, // id. transakce
                    uint64 *p_affected_rows,       // počet smazaných souborů
                    uint64 entry_id);              // ID souboru

/* vloží nový soubor (adresář) a vrátí jeho ID */
int db_insert_entry(db_transaction_t *transaction, // id. transakce
                    uint64 *p_entry_id,            // ID nového souboru
                    kivfs_file_type_t type,        // typ souboru
                    char *name,                    // název souboru
                    uint64 ctime,                  // čas vytvoření
                    uint64 user_id,                // ID vlastníka
                    kivfs_entry_t *parent_entry);  // rodičovský adresář

/* upraví adresář */
int db_update_directory(db_transaction_t *transaction, // id. transakce
                        uint64 *p_affected_rows,       // počet upravených
                                                       // adresářů
                        uint64 user_id,                // ID vlastníka
                        uint64 directory_id,           // ID adresáře
                        uint64 parent_id,              // ID rodičovského
                                                       // adresáře
                        char *name,                    // název adresáře
                        uint64 mtime,                  // čas změny
                        uint64 atime)                  // čas přístupu
```

Příloha B - Funkce poskytované databázovým konektorem

```
/* vykoná SQL dotaz a vrátí jeho výsledek */
int db_select(db_transaction_t *transaction, // identifikátor transakce
              void **p_result,               // výsledek dotazu
              char *format,                  // formát SQL dotazu
              ...);                          // parametry SQL dotazu

/* vloží záznam do databáze a vrátí jeho ID */
int db_insert(db_transaction_t *transaction, // id. transakce
              uint64 *p_inserted_id,         // ID vloženého záznamu
              char *format,                  // formát SQL příkazu
              ...);                          // parametry SQL příkazu

/* vykoná příkaz */
int db_execute(db_transaction_t *transaction, // id. transakce
               uint64 *p_affected_rows,       // počet ovlivněných záznamů
               char *format,                  // formát SQL příkazu
               ...);                          // parametry SQL příkazu

/* vloží záznam do databáze a vrátí jeho ID */
int db_real_insert(db_transaction_t *transaction, // id. transakce
                   uint64 *p_inserted_id,         // ID vloženého záznamu
                   char *query,                   // příkaz
                   uint64 length);                // délka příkazu

/* uvolní výsledek */
void db_free_result(void *result); // výsledek

/* vrátí další řádek z výsledku */
char **db_fetch_row(void *result); // výsledek

/* vrátí pole délek z aktuálního řádku výsledku */
unsigned long *db_fetch_lengths(void *result);  // výsledek

/* vrátí počet řádků z výsledku */
uint64 db_num_rows(void *result); // výsledek

/* ošetří vstupní řetězec */
int db_escape(db_transaction_t *transaction, // id transakce
              char **p_dst,                  // zdrojový řetězec
              char *src,                     // cílový řetězec
              uint64 length);                // délka zdojového řetězce

/* zahájí transakci */
int db_start_transaction(db_transaction_t **p_transaction); // id. zahájené
                                                            // transakce

/* zruší změny provedené v transakci */
int db_rollback_transaction(db_transaction_t **p_transaction); // id. transakce

/* potvrdí změny provedené v transakci */
int db_commit_transaction(db_transaction_t **p_transaction); // id. transakce
```

Příloha C - Relace mezi souborem a ACL

```
/* typ souboru */
typedef enum {
  FILE_TYPE_ANY            = 0,
  FILE_TYPE_DIRECTORY      = 1,
  FILE_TYPE_REGULAR_FILE   = 2,
} kivfs_file_type_t;


/* bity reprezentující jednotlivá práva
   na čtení, zápis a spuštění */
typedef enum {
  NONE_BIT    = 0,
  EXECUTE_BIT = 1,
  WRITE_BIT   = 2,
  READ_BIT    = 3
} kivfs_permission_bit_t;


/* práva */
typedef enum {
  NONE_PERMISSION               = 0,
  EXECUTE_PERMISSION            = 1,
  WRITE_PERMISSION              = 2,
  EXECUTE_WRITE_PERMISSION      = 3,
  READ_PERMISSION               = 4,
  READ_EXECUTE_PERMISSION       = 5,
  READ_WRITE_PERMISSION         = 6,
  READ_WRITE_EXECUTE_PERMISSION = 7
} kivfs_permission_t;


/* typy jmenných záznamů */
typedef enum {
  NAMED_USER  = 1,
  NAMED_GROUP = 2,

  DEFAULT_NAMED_USER  = 3,
  DEFAULT_NAMED_GROUP = 4,
} kivfs_named_acl_type_t;


/* datový typ role */
typedef struct {
  uint64 id;
  char   *name;
} kivfs_role_t;

/* funkce pro správu role */
kivfs_role_t *kivfs_role(uint64 id,
    char *name);
void kivfs_print_role(kivfs_t *role);
void kivfs_free_role(kivfs_t *role);
```

```
/* datový typ jmenný záznam */
typedef struct {
  kivfs_named_acl_type_t  type;
  kivfs_permission_t      permission;
  kivfs_role_t            *role;
} kivfs_named_acl_t;

/* funkce pro správu jmenného záznamu */
kivfs_named_acl_t *kivfs_named_acl(kivfs_named_acl_type_t type,
    kivfs_permission_t permission,
    kivfs_role_t *role);
void kivfs_print_named_acl(kivfs_t *named_acl);
void kivfs_free_named_acl(kivfs_t *named_acl);


/* datový typ ACL */
typedef struct {
  kivfs_permission_t owner;
  kivfs_permission_t group;
  kivfs_permission_t other;

  kivfs_permission_t mask;

  kivfs_list_t       *named_users;
  kivfs_list_t       *named_groups;

  kivfs_permission_t default_owner;
  kivfs_permission_t default_group;
  kivfs_permission_t default_other;
  kivfs_permission_t default_mask;

  kivfs_list_t       *default_named_users;
  kivfs_list_t       *default_named_groups;
} kivfs_acl_t;

/* funkce pro správu ACL */
kivfs_acl_t *kivfs_acl(kivfs_permission_t owner,
    kivfs_permission_t group,
    kivfs_permission_t other,
    kivfs_permission_t mask,
    kivfs_list_t *named_users,
    kivfs_list_t *named_groups,
    kivfs_permission_t default_owner,
    kivfs_permission_t default_group,
    kivfs_permission_t default_other,
    kivfs_permission_t default_mask,
    kivfs_list_t *default_named_users,
    kivfs_list_t *default_named_groups);
void kivfs_print_acl(kivfs_t *acl);
void kivfs_free_acl(kivfs_t *acl);


/* datový typ soubor */
typedef struct {
  kivfs_file_type_t type;
  char *name;
```

```
  char *owner;
  char *group;

  uint64 size;
  uint64 mtime;
  uint64 atime;
  uint64 version;
  uint64 read_hits;
  uint64 write_hits;

  kivfs_acl_t *acl;
} kivfs_file_t;

/* funkce pro správu souboru */
kivfs_file_t *kivfs_file(kivfs_file_type_t type,
    char *name,
    uint64 size,
    uint64 mtime,
    uint64 atime,
    uint64 version,
    uint64 read_hits,
    uint64 write_hits,
    char *owner,
    char *group,
    kivfs_acl_t *acl);
void kivfs_print_file(kivfs_t *file);
void kivfs_free_file(kivfs_t *file);

/* datový typ obecný souborový záznam */
typedef struct entry_st {
  uint64 id;
  uint64 owner_id;
  uint64 group_id;
  uint64 volume_id;

  uint64 read_locks;
  uint64 write_locks;

  struct entry_st *parent;
  kivfs_file_t    *file;
} kivfs_entry_t;

/* funkce pro správu souborového záznamu */
kivfs_entry_t *kivfs_entry(uint64 id,
    uint64 owner_id,
    uint64 group_id,
    uint64 volume_id,
    uint64 read_locks,
    uint64 write_locks,
    kivfs_entry_t *parent,
    kivfs_file_t *file);

void kivfs_print_entry(kivfs_t *entry);

void kivfs_free_entry(kivfs_t *entry);
```

Příloha D - Správa šifrovaných souborů

```
typedef struct {
  gcry_cipher_hd_t  cipher_handler; // cipher handler
  kivfs_boolean_t   editable,       // soubor otevřen pro editaci
                    changed;        // obsah souboru byl změněn
  FILE              *file;          // souborový proud
  uint64            writed;         // počet zapsaných bajtů
} encrypted_part_t;

/* zapisuje data do šifrovaného podsouboru */
int write_to_part(encrypted_part_t *encrypted_part, void *buffer,
    uint64 length);

/* čte data ze šiforvaného podsouboru */
int read_from_part(encrypted_part_t *encrypted_part, void *buffer,
    uint64 length);

/* otevírá šifrovaný podsoubor (pro čtení, zápis, čtení/zápis) */
int open_part(encrypted_part_t **p_encrypted_part, kivfs_replica_t *replica,
              kivfs_part_t *part, uint64 offset, kivfs_file_mode_t mode);

/* zavírá šifrovaný podsoubor */
int close_part(encrypted_part_t **p_encrypted_part,
               kivfs_replica_t *replica, kivfs_part_t *part);
```

Příloha E - Seznam příkazů klienta

Příkaz	Parametry	Popis testované funkce
ls	<path>	vypíše obsah adresáře
mkdir	<path>	vytvoří adresář
rmdir	<path>	smaže adresář
rm	<path>	smaže soubor
mv	<path><path>	přesune soubor
info	<path>	zjistí informace o souboru
hits	-	zjistí hodnotu čítačů přístupů k souboru
touch	<path>	změní časové známky souboru
chmod	<7777><path>	změní práva souboru
dchmod	<7777><path>	změní výchozí práva souboru
chown	<owner><path>	změní vlastníka souboru
chgrp	<group><path>	změní vlastnickou skupinu souboru
setfacl	<type><role> <permission><path>	nastaví ACL souboru
getfacl	<path>	zjistí ACL souboru
id	<user>	zjistí informace o uživateli
su	<user>	přepne uživatele
groupadd	<name>	přidá skupinu
groupdel	<name>	smaže skupinu
useradd	<user><group>	přidá uživatele
userdel	<name>	odebere uživatele
engroup	<user><group>	přidá uživatele do skupiny
ungroup	<user><group>	odebere uživatele ze skupiny
begin		zahájí transakcí
commit		potvrdí změny provedené v transakci
rollback		zahodí změny provedené v transakci
maxtime		zjistí maximální logickou časovou známku v logu
logs		vypíše všechny logy
df		vypíše stav svazků
quota	<volume><size>	nastaví kvótu svazku
get	<remote path><local path>	stáhne soubor
put	<local path><remote path>	nahraje soubor
rewrite	<local path><remote path>	přepíše soubor
append	<local path><remote path>	přidá soubor na konec souboru

Příloha F - Konfigurační soubor

```
[kivfs_global]
server_name=fs-1
log_directory=/tmp
debug=yes
max_clients=5000
max_queue=500
local_mode=yes

[sync]
port=30002
ip=127.0.0.1

[vfs_server]
ip=127.0.0.1
port=30003

[db_server]
ip=127.0.0.1
port=30004

[fs_server]
ip=127.0.0.1
port=30005

[database]
name=kivfs_db
user=kivfs
password=*****
host=localhost
port=3306
pooling=100

[storage]
root_directory=/mnt/kivfs
temporary_directory=/mnt/kivfs/temp
encryption_key=5b2d2840662c333b6e773a682b2a355739722d4d26534c3272665f5b3d
encrypt_storage=yes
hash_files=yes
blocksize=52428800
files_count=1000
deduplication_interval=324000
replication_interval=300
removing_interval=180
```