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

Diplomová práce

KIVFS - server

Plzeň 2017 Bc. Tomáš Weisheitel

Místo této strany bude zadání práce.

Prohlášení

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

V Plzni dne 27. června 2017

Bc. Tomáš Weisheitel

Abstract

The goal of this work is to study the existing distributed file system KIVFS and create a new distributed file system KIVFS2. This system will eliminate the shortcomings of the KIVFS system, especially the architecture and way of communication beetween the layers will be rework. These changes will lead to increased performance and stability of the newly implemented system.

The text is divided into five chapters. The first chapter outlines the issue and the main objectives of the work. The second chapter describes important mechanisms used in distributed file systems. The third chapter focuses on the design and implementation of KIVFS2. The next chapter describes test scenarios and their evaluation. The last fifth chapter summarizes the most important information and suggestions for further improvements.

Abstrakt

Cílem této práce je prostudovat existující distribuovaný souborový systém KIVFS a na základě získaných znalostí navrhnout nový distribuovaný souborový systém KIVFS2. Tento systém bude odstraňovat nedostatky systému KIVFS, především bude předělána architektura a způsob komunikace mezi jednotlivými vrstvami. Tyto změny povedou ke zvýšení výkonu a stability nově implementovaného systému.

Text je rozdělen do pěti kapitol. Kapitola první nastiňuje danou problematiku a hlavní cíle práce. Ve druhé kapitole jsou popsány důležité mechanizmy používané v distribuovaných souborových systémech. Třetí kapitola je zaměřena na návrh a implementaci systému KIVFS2. V další kapitole jsou popsány testovací scénáře a jejich zhodnocení. Poslední pátá kapitola shrnuje nejdůležitější získané informace a návrhy dalšího vylepšení.

Obsah

1 Úvod

Tato práce pojednává o distribuovaném souborovém systému, který se nazývá KIVFS[25] a je vytvářen od roku 2008 na Katedře informatiky a výpočetní techniky Západočeské univerzity v Plzni.

V dnešní době stále roste množství dat, které je třeba efektivně uchovávat. Z tohoto důvodu jsou stále více vyhledávány systémy, které umožní bezpečně toto množství dat uchovávat, aniž by došlo k jejich ztrátě nebo odcizení. Nezbytnou součástí je umožnění přístupu k uloženým datům, téměř odkudkoliv a z jakéhokoliv zařízení. Přesně takovým systémem je distribuovaný souborový systém. Důvodem vzniku KIVFS, byly nedostatky současných distribuovaných souborových systémů, způsobené rostoucími nároky. KIVFS přináší nové vlastnosti, které nenabízí žádný z používaných systémů. Nové vlastnosti jako je dynamický routing nebo multi-master online replikace.

KIVFS od doby vzniku vyvíjelo mnoho týmu a jednotlivců, což vedlo k tomu, že jeho momentální stav je ne udržovatelný. Cílem této práce je tedy reimplementace serverové části systému KIVFS na základě znalostí získaných z předchozích prací. Vzhledem k těmto znalostem bude předělána především architektura a přidán nový způsob komunikace mezi jednotlivými vrstvami. Tento refaktoring povede ke zvýšení výkonu a stability.

Práce je členěna do několika kapitol. Kapitola 2 obsahuje teoretický popis technologií, které jsou používány v distribučních souborových systémech včetně KIVFS. Kapitola 3 je zaměřena na implementaci technologií v nově navrženého systému. V kapitole 4 jsou popsány provedené testy a zhodnocení jejich výsledků oproti testům prováděným v minulosti. Kapitola 5 uzavírá celou práci zhodnocením vytvořeného systému a provedených testů.

2 Teoretická část

Teoretická část se věnuje úvodu do problematiky distribuovaných souborových systémů. Dále jsou zde uvedeny existující souborové systémy, včetně KIVFS a návrh nového systému KIVFS2.

2.1 Centralizovaný systém

Jedná se o systém, kde veškerou činnost řídí jeden centrální uzel. Jedině centrální uzel má právo provádět změny. Ostatní uzly mohou provedené změny pouze číst. V centralizovaných systém je velice snadná implementace globálního času, protože existuje pouze jeden uzel, skrz který prochází veškerý tok požadavků. Globální čas se tedy nemusí synchronizovat s ostatními uzly, jak tomu je v systémech distribuovaných. Nevýhodou tohoto systému je velká zátěž na centrální uzel. Při výpadku řídícího uzlu je nezbytné pomocí algoritmu vybrat uzel, který převezme roli centrálního uzlu v systému.

2.2 Distribuovaný systém

Existuje několik definic distribuovaných systémů. Podle odborné literatury [1] je distribuovaný systém definován jako kolekce nezávislých systémů, které se uživateli jeví jako jeden funkční celek. To znamená, že jednotlivé komponenty musí mezi sebou navzájem spolupracovat. Principy, jak spolu jednotlivé uzly komunikují je uživateli skryto.

Distribuované systémy jsou konstruovány za účelem sdílení prostředků, transparentního přístupu a řešení konkurenčního přístupu. Základní vlastnosti jsou:

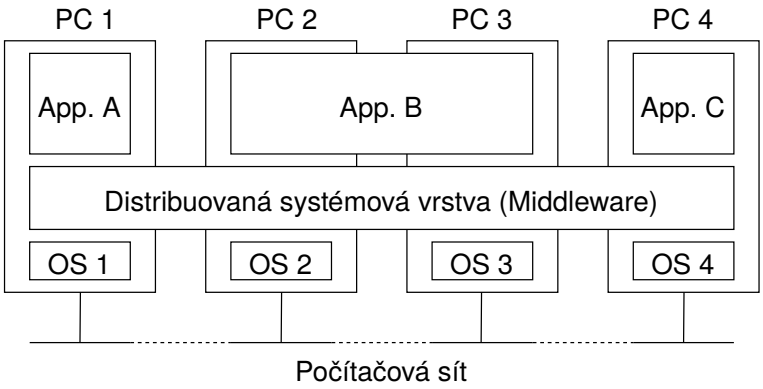
- **Neexistuje centrální uzel** – to znamená, že neexistuje uzel, který by zpracovával každý požadavek. Celý systém je řízený distribuovaným algoritmem.
- **Neobsahují sdílenou paměť** – všechny uzly mezi sebou komunikují pomocí zpráv. Lze tedy implementovat předávání dat, která mohou být uložena na jednom nebo více uzlech.
- **Škálovatelnost** – velká výhoda distribuovaných systémů spočívá ve snadné horizontální škálovatelnosti celého systému. Každý uzel je au-

tonomní. To znamená, že lze přidat a ubrat uzel, aniž by došlo k nestabilitě celého systému. Z toho vyplývá další výhoda. Systém dokáže fungovat i v případě, že některý uzel havaruje.

- **Heterogenita** – každý uzel může být tvořen počítačem s různým kódem, různou architekturou nebo různým operačním systémem.

Aby bylo možné zachovat heterogenitu, a přesto se dívat na systém jako na jeden funkční systém, distribuované systémy jsou často organizovány do logické vrstvy, která leží mezi vyšší vrstvou uživatelů a aplikací a vrstvou samotného operačního systému. Tato vrstva se nazývá middleware, viz obr. 2.1.

V reálných systémech je často na některé vlastnosti kladen větší důraz než na jiné. Vše záleží na účelu, pro který je daný distribuovaný systém konstruován. Speciální podskupinu distribuovaných systémů tvoří distribuované souborové systémy, kterým je věnována kapitola 2.3.



Obrázek 2.1: Distribuovaný systém organizovaný jako middleware. Middleware poskytuje každé aplikaci stejný interface.

2.3 Distribuované souborové systémy

Distribuované souborové systémy jsou podmnožinou distribuovaných systémů. Jedná se o souborový systém, ve kterém jsou servery, klienti a uložiště rozptýleny mezi uzly distribuovaného systému. Veškerá komunikace distribuovaného souborového systému je prováděna napříč počítačovou sítí. Na rozdíl od jednoho centralizovaného uložiště, obsahuje distribuovaný souborový systém n nezávislých uložišť.[8]

Pro správnou funkčnost využívá distribuovaný souborový systém těchto mechanizmů:

- **Šifrování** - ochrana před odcizením nebo poškozením přenášených dat
- **Autentizace** – ověření totožnosti uživatele
- **Autorizace** - ověření, jestli má uživatel povolení k přístupu
- **Replikace** – udržování dat ve více kopiích
- **Konzistence** – výsledek, po provedení požadavku, musí být na všech replikách stejný
- **Distribuované transakce** – skupina operací, která vede ke změně z jednoho konzistentního stavu do druhého
- **Synchronizace** – proces komunikace, ohledně stejného provedení daného požadavku

Klient s distribuovaným souborovým systémem často komunikuje skrz počítačovou síť, což není bezpečné prostředí. Proto by každá komunikace měla být šifrována. Veškerá komunikace mezi klientem a distribuovaným souborovým systémem by tedy měla být odolná proti potencionálním útočníkům.

2.3.1 Bezpečnost

Jedna z klíčových součástí každého distribuovaného souborového systému je ochrana dat. Protože se pohybujeme v heterogenním prostředí, musí systém zajistit co největší ochranu dat před možným napadením. V distribuovaném souborovém systému jsou data chráněna omezením přístupu uživatele k datům. Toto omezení je zajištěno několika způsoby:

- **Šifrování komunikace** – Šifrovaná spojení zajistí ochranu před odcizením nebo poškozením přenášených dat.
- **Autentizace uživatele** – Než je uživateli umožněn přístup do systému, musí prokázat svou identitu. Po úspěšném ověření je mu povolen přístup.
- **Autorizace uživatele** – Uživatel by neměl mít přístup k datům, které mu přímo nepatří nebo nevlastní práva pro manipulaci s cizími daty. Autorizace zajišťuje ověření, jestli daný uživatel má právo manipulovat se zvolenými daty.

Autentizace

patří k nejdůležitějším úkolům v oblasti bezpečnosti. Jak již bylo zmíněno, jedná se o ověření totožnosti uživatele. Na základě tohoto procesu je systém schopen ověřit, zda je uživatel opravdu ten, za kterého se vydává. Při autentizaci dochází k ověření na základě informace, kterou znají jak server, tak uživatel. Jedna z nejčastějších metod je ověření na základě jména a hesla. Další možností je použití tokenu. Jedná se o elektronický klíč používaný pro přístup. Pro zvýšení bezpečnosti lze tento klíč ještě zabezpečit heslem. Poslední možností je metoda biometrie. V tomto případě je pro autentizaci použit například otisk prstu nebo scan oka.

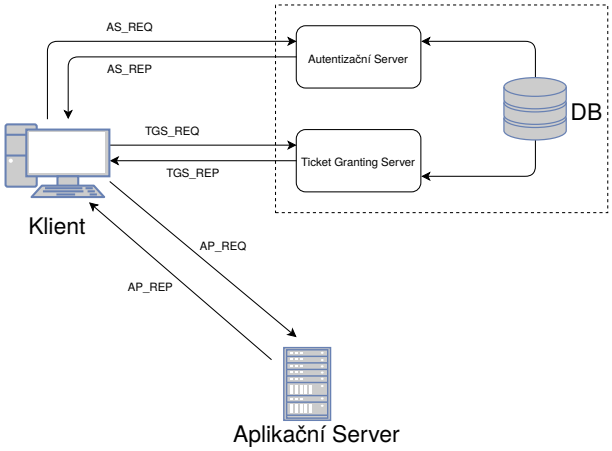
V distribuovaném souborovém systému lze využít například autentizačního protokolu LDAP[17], NTLM[12] nebo Kerberos[11]. Pro KIVFS byl vybrán protokol Kerberos, protože nikdy nepřenáší heslo skrz síť na rozdíl od LDAP a nahrazuje starší protokol NTLM.

Kerberos je síťový autentizační protokol. Byl navržen tak, aby poskytoval silnou autentizaci pro ověření klienta na serveru (a naopak), za použití symetrické kryptografie, v nezabezpečeném prostředí. Po úspěšné autentizaci lze tímto protokolem šifrovat i celou komunikaci. Největší výhodou protokolu Kerberos je, že heslo není nikdy přenášeno skrz síť.

Kerberos pracuje na principu ticketů. Je založen na Needham-Schroeder Symmetric Key Protocol. Tento protokol využívá třetí strany, která se nazývá Key Distribution Center (KDC). KDC se sestává ze dvou částí: Authentication Server (AS) a Ticket Granting Service (TGS). Klient se nejprve ověří vůči AS a obdrží ticket. Následně kontaktuje TGS a pomocí ticketu prokáže svou totožnost a požádá o přístup ke službě. Od TGS, po úspěšném ověření, obdrží další ticket, kterým se prokáže u poskytovatele služby. Komunikace protokolu Kerberos je ukázána na obr. 2.2.

Autorizace

je proces, při kterém dochází k ověření, jestli má daný uživatel přístup k požadovaným datům. Toto ověření probíhá například na základě vyhledání v seznamu. Mnoho distribuovaných souborových systémů používá Access control list (ACL). Jedná se o seznam oprávnění připojený k nějakému objektu. ACL určuje, kdo má přístup k danému objektu a jaké operace s ním může provádět. Tomuto procesu často předchází proces autentizace.



Obrázek 2.2: Ukázka komunikace protokolu Kerberos.

2.3.2 Rozdělení dat

Distribuovaný souborový systém, jako je například KIVFS, uchovává dva typy dat. Tím prvním typem jsou fyzická data, jakou jsou soubory. Druhý typ jsou metadata. Jedná se o data, která popisují některá jiná data. Například se může jednat o popis adresářové struktury, uložení fyzických souborů nebo informace o serverech kde se uchovávají replikace.

Data

Fyzická data jsou nejčastěji uchovávána na souborovém systému serveru, na kterém běží server distribuovaný souborový systém. K nejznámějším souborovým systémům patří například NTFS, FAT nebo ext. Další možností je uchovávání dat v databázi. Tato možnost není výhodná hlavně z pohledu paměťové náročnosti, čímž by vzrostla velikost dané databáze a vedlo by to ke komplikaci s replikacemi.

Metadata

Jedna z možností jak distribuovaný souborový systém může uchovávat adresářovou strukturu, je rovněž na souborovém systému daného serveru. Efektivnější možností zde je uchovávat adresářovou strukturu formou metadat. Nejvhodnějším uložištěm pro metadata je databázový systém. Použití metadat uložených v databázi, zvyšuje rychlost systému a poskytuje možnost replikovat databázi na všechny servery, kvůli její malé paměťové náročnosti.

Existující databázové systémy

DBMS jsou programy umožňující uživateli vytvářet a spravovat databázi. DBMS by mělo napomáhat k navrhování, implementování, udržování a šíření databáze mezi různými uživateli nebo aplikacemi.

- **Návrh** – jedná se o specifikaci, jaká data budou v databázi uložena a jaké budou jejich datové typy.
- **Implementace** – vytvoření databáze, předělení na jaké médium se bude ukládat.
- **Manipulace** – funkce pro manipulaci s daty. Jedná se o funkce vytváření, vkládání, zobrazení a mazání dat.
- **Sdílení** – umožňuje přístup do databáze různým uživatelům nebo programům.

Dále jsou uvedeny informace o dnes nejvíce používaných databázových systémech.

MySQL MySQL[5] je relační databázový systém typu klient / server vytvořen švédskou společnostní MySQL AB. Nabízí dvojí licencování - bezplatnou licenci GPL a nebo placenou komerční licenci. Díky modelu klient-server je možné s databází pracovat z jakéhokoliv zařízení, které obsahuje MySQL klienta. Tento systém je snadno přenositelný. MySQL je optimalizováno především na rychlost, a to i za cenu zjednodušení některých funkcí (zálohování, pohledy, triggery). Poskytuje také tzv. MySQL konektory a API funkce, které umožňují připojení z mnoha prostředí (např. C/C++, Java, PHP). Nejčastěji se používá pro webové aplikace.

PostgresSQL PostgresSQL[29] je objektově-relační databázový systém. Jedná se o open source, který je vytvářen komunitou a je dostupný pod BSD a MIT licencemi. Tyto licence prakticky umožňují uživateli dělat cokoliv, včetně úprav zdrojového kódu. Předchůdce tohoto systému byl systém Ingres, vyvíjený na kalifornské univerzitě. Tento systém klade velký důraz na co nejkomplexnější funkcionalitu. Je velmi snadno rozšiřitelný a platformě nezávislý. Využívá vlastní procedurální jazyk PL/pgSQL připomínající PL/SQL známý z Oracle. PostgresSQL umožňuje běh uložených procedur napsaný v několika programovacích jazycích (Perl, Python, C).

Oracle Jedná se komerční, multiplatformní DBMS s pokročilou funkcionalitou, která přesahuje výše uvedené systémy. Oracle[3] byl navržen pro podnikové systémy. Tento systém klade velký důraz na robustnost a funkcionalitu. Podporuje veškeré dnes požadované vlastnosti (transakce, zámky, triggery, pohledy apod.). Stejně jako předchozí systémy se jedná o model klient / server. I přesto, že se jedná o placený produkt, Oracle je k dispozici i v bezplatné verzi. Tato verze je však výkonnostně omezena (velikost databáze, využití procesoru, velikost pamětí) bez ohledu na výkon serveru.

MS SQL Relační databázový systém od společnosti Microsoft[26]. Používá dotazovací jazyk Transact-SQL. Jedná se o rozšíření standartního jazyka SQL od společnosti Microsoft. Tento systém je dostupný pouze pro platformy Windows. Podporuje celou řadu funkcionality stejně jako Oracle. Podle žebříčku NIST se jedná o nejméně zranitelnou databázi. Produkt je dostupný ve dvou verzích. V plné verzi se jedná o komerční placený produkt. Bezplatná verze (Express Edition) je opět výkonnostně omezena.

Databázový systém v KIVFS2 Pro KIVFS2 je nezbytné, aby databázový systém dovedl provádět read-write replikace. Toto kritérium nesplňuje ani jeden z výše uvedených systémů. Z tohoto důvodu je nutné zavést další mechanismus pro synchronizaci. Jako databázový systém byl vybrán MySQL, z důvodu rychlosti práce s daty.

2.3.3 Replikace

V rámci distribuovaných systémů, se data často uchovávají ve více kopiích, tzv. replikách. Replikace[24] jsou klíčem ke zlepšení distribuovaného systému. Hlavní motivací pro replikaci dat jsou:

- **Zvýšení výkonu** – Díky replikacím je možné provádět rovnoměrné vyvážení zátěže v celém systému. Nedochází tedy ke zpomalení některých více využívaných serverů. Protože data mohou být poskytována z různých zdrojů, lze poskytovat uživateli taková data, která jsou k němu nejblíže.
- **Zvýšení dostupnosti** – Pokud nastane výpad některého z uzlů, existuje jiný uzel, který dokáže zastoupit právě nefunkční uzel. Uživatel tedy ani nepozná, že k nějakému výpadku v rámci systému došlo.

K replikacím se váží i nevýhody, které je třeba vyřešit. Je zapotřebí udržovat konzistenci existujících replik. To znamená, že pokud měníme data

na jedné replice, musí být tyto změny propagovány i na ostatní repliky ve stejném pořadí.

Typy replikací

Replikace lze rozdělit podle toho, jak zacházejí s replikami na dva typy:

- **Read-only replikace** – Existuje pouze jeden primární uzel (master), který zpracovává veškeré operace zápisu. Ostatní uzly se nazývají vedlejší (slaves). Tento typ replikace lze nazývat master-slave replikace. Po tom, co proběhne zápis na primární uzel, jsou změny propagovány na uzly vedlejší. Vedlejší uzly slouží pouze ke čtení. V případě, že vypadne primární uzel, existuje mechanismus na zvolení nového primárního uzlu z vedlejších uzlů. Tento nově zvolený uzel se stává novým primárním uzlem. Nevýhodou tohoto typu je velká zátěž na primární uzel. Read-only replikace využívá například distribuovaný souborový systém AFS.
- **Read-write replikace** – V typu read-write, nebo také master-master, jsou si všechny uzly rovny. Každý z nich má možnost, jak zapisovat, tak číst. Po operaci zápisu jsou změny odeslány na ostatní uzly, na kterých dojde k zápisu provedených změn. Výpadek některého z uzlů nemá žádný vliv na funkčnost ostatních uzlů. Tento typ musí mít implementován mechanizmus, který bude řešit konflikty při změně. Read-write replikace jsou použity v distribuovaném souborovém systému KIVFS.

Z pohledu výkonosti jsou read-write replikace vhodnější pro použití v distribuovaných souborových systémech, i přes to, že je zapotřebí vetší režii při řešení konfliktů.

2.3.4 Konzistence

Při změně dat, na jedné replice, je nutné, aby byl výsledek stejný i na ostatních replikách. To znamená, že pokud měníme data na jedné replice, musí být tyto změny propagovány i na ostatní repliky, a to ve stejném pořadí, podle zvolených logických hodin. Právě podle způsobu propagace rozlišujeme několik modelů konzistence. Dva základní pohledy na konzistenci jsou data-centric modely a client-centric modely. Model data-centric je zaměřen na data. Existuje několik procesů pracujících nad jednou skupinou dat. Naproti tomu model client-centric je zaměřen na uživatele. To znamená, že existuje jeden proces ale několik skupin dat. Další možnost dělení je z hlediska

propagace změn dat. Níže uvedené modely konzistence popisují pravidla, která je nezbytné dodržovat k zachování konzistence.

Striktní konzistence

Jedná se o konzistenci, která je ze všech nejsilnější. Tuto konzistenci běžně implementují centralizované systémy a lze ji definovat následovně. Jakékoliv čtení dat x vrátí hodnotu uloženou při posledním zápisu dat x. Tento model předpokládá existenci globálního času, aby bylo možné určit pořadí jednotlivých akcí.

Sekvenční konzistence

Sekvenční konzistence předpokládá, že operace zápisu na všech uzlech jsou řazeny ve stejném pořadí. To znamená, že operace nemusí být provedeny ve stejném čase. Právě toto zpoždění může způsobovat nekonzistenci. Lze implementovat pomocí protokolu ABCAST[6].

Příčinná konzistence

Příčinná konzistence představuje model, kde operace, které jsou na sebe vázány, musí být viděny všemi uzly ve stejném pořadí. Konkurenční zápisy mohou být viděny různými uzly v různém pořadí. Je zapotřebí rozlišovat operace, které jsou na sebe vázany a udržovat graf závislostí zápisů a čtení.

FIFO konzistence

Model, u kterého jsou všechny zápisy provedeny jedním uzlem, jsou ostatními procesy viděny v tom samém pořadí, ve kterém byly prováděny. Zápisy různých uzlů mohou být viděny různými procesy v různém pořadí. FIFO konzistence je snadno implementovatelná, jediné co je potřeba dodržet, je pořadí zápisů jednoho zdroje

Slabá konzistence

Slabá konzistence, na rozdíl od předchozích modelů, využívá nástrojů, které umožňují přenášet ostatním uzlům pouze konečný výsledek poslední operace. Tyto nástroje se nazývají synchronizační proměnné. Zavedením těchto nástrojů se zmenšuje počet přenesených zpráv, a tím i zatížení sítě. Přístup k synchronizačním proměnným je sekvenčně konzistentní a lze ho řešit například pomocí Lamportova algoritmu.

Uvolňovací konzistence

U této konzistence existuje kritická sekce. Všechny změny jsou propagovány při opuštění kritické sekce. Před opuštěním této sekce musí být ukončeny všechny předchozí požadavky. Přístup do kritické sekce je obdobný jako v případě paralelního programování a musí být FIFO konzistentní.

Přístupová konzistence

Jedná se o podobný model jako v případě uvolňovací konzistence. Hlavním rozdílem je, že přístup k synchronizační proměnné není povolen, dokud nebyly provedeny všechny změny.

Konzistence v KIVFS2

Nejvhodnější konzistencí pro KIVFS2 by byla striktní konzistence. Dosažení této konzistence by bylo příliš nákladné a složité. Z toho důvodu je v KIVFS2 použita konzistence sekvenční.

2.3.5 Distribuované transakce

Transakce je skupina operací, které převedou systém z jednoho konzistentního stavu do druhého. Transakce musí splňovat ACID kritéria. Jedná se o následující vlastnosti:

- **Atomicita** – Provede se buďto celá série operací nebo je systém ponechán v původním stavu.
- **Konzistence** – Každá transakce převede systém z jednoho konzistentního stavu do druhého.
- **Izolovanost** – Data měněná transakcí jsou ostatním uživatelům až do úspěšného ukončení viditelná v původní podobě.
- **Trvalost** - Změny jsou trvale uloženy.

Při vykonávání transakce v distribuovaných systémech, je nezbytné, aby mezi sebou uzly silně komunikovali. Výsledek transakce je synchronní a musí se projevit na všech uzlech nebo nikde. Každý uzel obsahuje frontu požadavků, které jsou seřazeny na základě logických hodin a čekají na vyřízení. Uzel, který požadavek přijal, se nazývá koordinátor. Koordinátor řídí průběh zpracování dané transakce a rozhoduje, jestli byla úspěšně provedena nebo nikoliv. Toto rozhodnutí realizuje na základě hlasování. Existují algoritmy, které

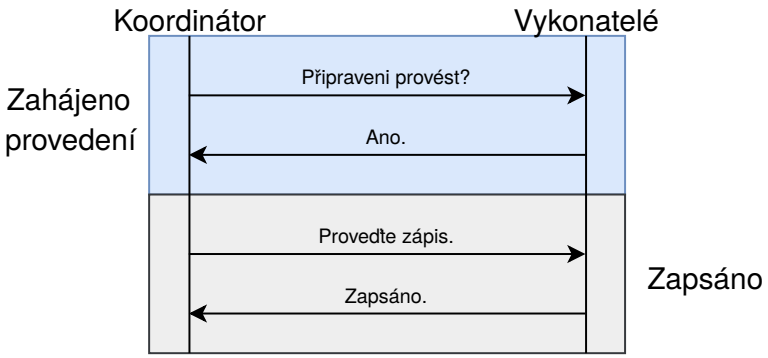
zajišťují spolehlivé provedení transakce. Jedná se o dvoufázové a třífázové potvrzování.

Dvoufázové potvrzování

Jedná se o algoritmus zajišťující globální integritu systému. Tento algoritmu probíhá ve dvou fázích, viz obr. 2.3. Celý průběh řídí koordinátor a rozhoduje o úspěchu respektive neúspěchu provedení transakce. V první fázi žádá koordinátor ostatní uzly, aby potvrdili, jestli jsou připraveni vykonat danou transakci. Uzly odpovídají buďto pozitivně nebo negativně.

Ve druhé fázi koordinátor rozhodne na základě přijatých odpovědí o dalším průběhu. Pokud jsou všechny odpovědi kladné, všem uzlům je odeslána informace že mají danou transakci provést. Pokud některý z účastníků odpověděl záporně, všem uzlům se odesílá informace o zrušení transakce. Nakonec je transakce buďto dokončena nebo vrácena do původního stavu.

Dvoufázové potvrzování je blokující operace. To zapříčiňuje zablokování všech uzlů včetně koordinátora. Tento nedostat je vyřešen v třífázovém potvrzování.



Obrázek 2.3: Dvoufázové provedení.

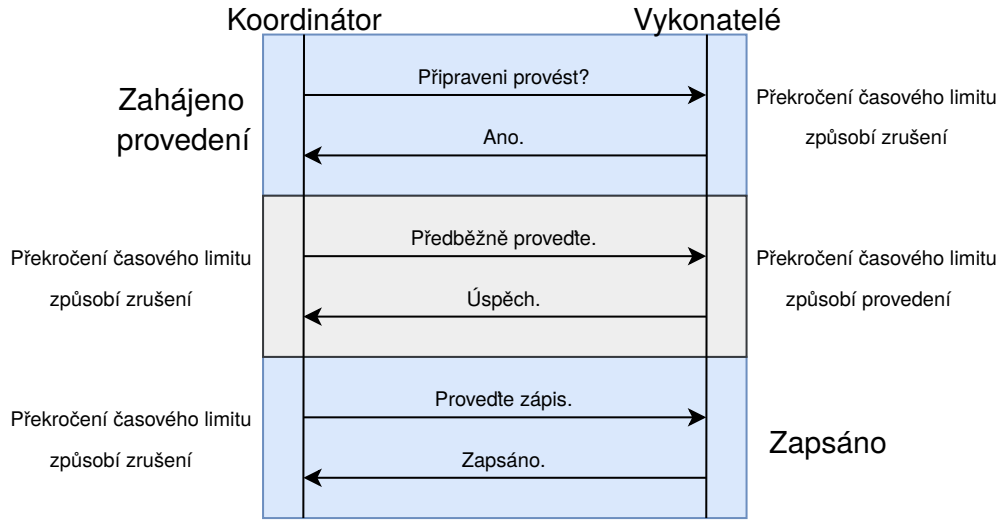
Třífázové potvrzování

Tento algoritmus vychází z algoritmu dvoufázového potvrzování. Na rozdíl od něj, ale odstraňuje jeho hlavní nedostatek tím, že zavadí časové limity, viz obr. 2.4.

V první fázi, stejně jako u dvoufázového potvrzování, žádá koordinátor všechny ostatní uzly o potvrzení připravenosti k provedení dané transakce. Uzly opět odpovídají pozitivně nebo negativně. Pokud uzly neobdrží žádost do konce časového limitu, dojde ke zrušení transakce.

Ve druhé fázi rozhodne koordinátor o předběžném zápisu na základě přijatých odpovědí. Opět platí, že pokud některý z uzlů odpoví záporně, celá transakce se ruší. Pokud koordinátor neobdrží včas odpověď, prohlásí transakci za neúspěšnou a odešle uzlům informaci o zrušení. Pokud jsou všechny potvrzení kladná, je odeslána žádost o předběžné provedení. U uzlů platí, že pokud dojde k překročení časového limitu, pokusí se transakci provést.

Ve třetí fázi požaduje koordinátor potvrzení zápisu. Pokud u všech uzlů proběhlo předběžné provedení úspěšně, je odeslána zpráva o potvrzení zápisu. Překročení časového limitu zapříčiní zrušení transakce.



Obrázek 2.4: Třífázové provedení.

Na rozdíl od dvoufázového potvrzování jsou zde definované časové limity a automatická reakce systému na jejich překročení.

Distribuované transakce v KIVFS2

V KIVFS2 je použita modifikovaná metoda dvoufázového potvrzování. Tato metoda je rozšířena o hlasování, kdy hlas odpojeného uzlu je brán jako záporný. Pokud uzel získá nadpoloviční počet hlasů, je operace provedena.

2.3.6 Synchronizace požadavku

Kapitola 2.3.5 řešila průběh jedné transakce, než k ní ale dojde, je třeba zajistit pořadí konkurenčních transakcí. V distribuovaných systémech, kde počet aktérů v systému může být n, musíme často řešit problém s konkurenčním přístupem. Je tedy nezbytné zajistit pořadí požadavků, aby bylo možné rozhodnout, který z požadavků bude nejprve proveden. V reálném

světe se
k určení pořadí využívají fyzické hodiny. V distribuovaných systémech, jak již bylo řečeno, neexistuje globální čas. Při použití lokálních hodin na každém uzlu, nastává problém s jejich synchronizací a přesností jejich měřením. Musí být zaveden jiný systém pro určování času a synchronizaci, který eliminuje nedostatky lokálních hodin. Pro tyto potřeby lze použít logické hodiny.

Fyzické hodiny

Jedná se o základní model, který se v distribuovaných systémech využívá např. v autentizaci a lze podle něj určovat také platnost certifikátů. Aby bylo možné synchronizovat požadavky podle lokálních hodin každého ze systému, je nezbytné na všech uzlech nastavit hodiny na stejný čas. Nevýhodou je, že jednotlivé časovače nemají stejnou frekvenci tiku. To znamená, že by docházelo k časté desynchronizaci a bylo by tedy nezbytné synchronizaci pořád opakovat.

NTP Pro synchronizaci fyzických hodin je používán Network Time Protokol (NTP)[28]. Jedná se o rozsáhle používaný protokol pro synchronizace lokálních hodin a vybraných serverových hodin. Přesnost NTP na LAN sítích je menší než milisekunda a na WAN sítích něco okolo pár milisekund. NTP je konfigurováno tak, aby používalo několik redundantních serverů a rozsáhlou síť cest, k dosažení co možná největší přesnosti a spolehlivosti.

Logické hodiny

Pro synchronizaci požadavku je podstatné v jakém pořadí se události mají provést. Není potřeba znát konkrétní čas události, ale jejich pořadí vůči ostatním. Lze tedy místo reálných hodin použít systém, který bude jednotlivým událostem přidělovat pořadová čísla. Musí tedy existovat jednoznačný algoritmus pro toto přidělování těchto čísel napříč distribuovaným systémem. Řazení lokálních událostí je snadné a lze k tomu použít primitiv známých z paralelních systémů (např. použití semaforů). V distribuovaných systémech jsou pro tyto účely implementovány logické hodiny. Synchronizace logických hodin je realizována za pomocí zasílání zpráv. Každý uzel má vlastní logické hodiny, které jsou se synchronizovány, a tím vzniká globální logický čas.

Lamportovy hodiny Nejjednodušším modelem logických hodin jsou Lampotovy hodiny. Obsahují pouze jedno bezrozměrné číslo, kterým jsou číslovány všechny operace. Při komunikaci mezi uzly, dochází k porovnání hodin a případné aktualizaci na vyšší hodnotu. Vymyslel je americký matematik

Lesley Lamport v roce 1978. Jejich největší výhodou je snadná implementace. Velkou nevýhodou však je nemožnost rozhodnout o pořadí konkurenčních operací, kde nedochází ke komunikaci. Je to z toho důvodu, že k synchronizaci dochází pouze při komunikaci mezi uzly.

Vektorové hodiny Vektorové hodiny vychází z modelu skalárních hodin a mají větší přesnost. Na rozdíl od Lamportových hodin, kde čítač byl tvořen jedním číslem, jsou vektorové hodiny tvořeny vektorem. Velikost tohoto vektoru je dána celkovým počtem uzlů. Každý prvek vektoru, reprezentuje jeden uzel. Opět platí, že synchronizace probíhá při komunikaci mezi uzly. S každou zprávou je odeslána i hodnota vektoru. Výhodou je opět jednoduchá implementace tohoto algoritmu. Další výhodou je možnost určit v jakém pořadí byli některé operace vykonány. Nevýhodou je, že nelze přesně určit, která ze souběžných operací nastala dříve.

Maticové hodiny Maticové hodiny rozšiřují hodiny vektorové. Logické hodiny v tomto modelu jsou reprezentovány maticí $n \times n$, nezáporných čísel. Tato matice reprezentuje pohled na globální logický čas v systému. Každý uzel U_i spravuje vlastní matici $m_i[1..n][1..n]$. Jednotlivé prvky matice reprezentují:

- $m_i[i : i]$ lokální události, čítač operací které se udály lokálně
- $m_i[i : j]$ poslední známý čítač lokálních hodin procesu $U_i(m_j[j : j])$
- $m_i[j : k]$ pohled na vztah lokálních hodin procesů $U_i(m_j[j : j])$ a $U_k(m_k[k : k])$

Každá zpráva obsahuje hodnotu maticových hodin, která je aktualizována při komunikaci mezi uzly. Maticové hodiny jsou striktně konzistentní a je tedy možné vždy rozhodnout, která operace proběhla dříve. Jedná se o velkou výhodu oproti předchozím modelům. Nevýhodou je náročnější implementace.

Synchronizace požadavku v KIVFS2 Pro synchronizaci požadavku v systému KIVFS2 je použito Lamportových hodin. Hlavním důvodem použití Lamportových hodin je rychlost algoritmu při použití tohoto typu hodin. Dalšími důvody jsou snadná implementace a vyhovující vlastnosti pro použití v systému KIVFS2.

2.3.7 Dynamické směrování požadavku

Dynamické směrování požadavku, umožňuje určit cestu kudy se požadavek dostane k cílovému uzlu. Na základě zavedení směrování, lze docílit vyšší rychlosti pro přenos dat nebo požadavků.

V distribuovaných systémech může mít každý uzel jinou geografickou polohu. Tento fakt může být komplikací, protože každý uzel musí být propojený, s ostatními uzly v systému, skrz nějakou linku. Každá z těchto linek může mít rozdílnou kvalitu. Lze tedy hodnotit kvalitu dané linky na základě zvolené metriky, která může být kombinací několika parametrů. Pro distribuovaný souborový systém je žádoucí, aby linka měla vysokou rychlost přenosu a malou odezvu. Pokud by se například odezva zvyšovala v čase, mohlo by nastat zpomalení celého systému. To vede k potřebě hledat co nejrychlejší cestu pro přenos dat. Tento problém lze vyřešit tím, že se celá fyzická síť překryje logickou sítí, která bude využívat co nejrychlejších linek. Na základě vytvořené logické sítě, lze provádět směrování požadavků.

Logickou síť lze vytvořit algoritmem pro hledání nejkratší cesty. Je totiž možné si distribuovaný systém (tedy fyzickou síť) představit jako ohodnocený a orientovaný graf. Kde uzly tvoří vrcholy v grafu a linky tvoří hrany. Každá hrana je ohodnocena na základě zvolené metriky. Je vhodné uvažovat orientované hrany, protože linky mohou být asymetrické (může se lišit jejich přenosová rychlost pro download a upload). Pro vyhledání nejkratší cesty lze využít algoritmy, které existují v teorii grafů. Těmito algoritmy jsou Dijkstrův algoritmus[36], Floyd-Warshall algoritmus[14] a Bellman-Ford algoritmus[13].

Dijkstrův algoritmus

Jedná se o algoritmus pro hledání nejkratších cest z počátečního uzlu do všech ostatních v kladně ohodnoceném grafu, který popsal nizozemský informatik Edsger Dijkstra. Algoritmus pracuje s množinou vrcholů V, hran E a počátečním vrcholem s. Dále využívá množinu navštívených vrcholů T a nenavštívených vrcholů N. Pro každý vrchol$v$ z V je hodnota $d[v]$ nejkratší délkou cesty z vrcholu s do v. Na počátku jsou všechny hodnoty $d[v]$ nastaveny na nekonečno, kromě počátečního vrcholu s. Dále prochází jednotlivé vrcholy z N a hledá nejkratší vzdálenost mezi všemi sousedy. Pokud je cesta přes nový vrchol kratší, nastaví se $d[v]$ na tuto hodnotu. Vrcholy z množiny N se po navštívení přesouvají do množiny T. Algoritmus končí, když je množina N prázdná. Složitost tohoto algoritmu je v nejhorším případě $O(n^2)$.

Floyd-Warshall algoritmus

Algoritmus, který hledá nejkratší cesty mezi všemi vrcholy grafu. V každém průchodu zpřesňuje odhad nejkratší cesty, doku nenalezne nejkratší cestu. V nejhorším případě se počet cyklů rovná počtu uzlů. Algoritmus pracuje s maticí $N \times N$, kde N je počet uzlů. V každém cyklu nastavuje hodnoty matice na kratší vzdálenost, tím že zkouší nalézt cestu přes větší počet vrcholů. Jeho časová složitost je $O(n^3)$ a paměťová náročnost je V^3.

Bellman-Ford algoritmus

Oproti předchozím algoritmům, tento dokáže pracovat i se záporně ohodnocenými hranami. Nejkratší cestu hledá z vrcholu s do všech ostatních vrcholů, obdobně jako Dijkstrův algortimus. Algoritmus nejprve nastavý počáteční hodnoty. Dále se zkoumá všechny hrany a zjišťuje, jestli neexistuje kratší cesta. Následně vyhledává záporné cykly. Pokud graf obsahuje záporné cykly, je možné opakováním jeho průchodů získávat stále kratší cestu. Časová složitost tohoto algoritmu je $O(|E||V|)$ a paměťová složitost je $2V^2$.

Směrování požadavku v KIVFS2

Na základě provedeného testování v diplomové práci od pana Jindřicha Skupy[37] byl pro hledání nejkratších cest vybrán Dijkstrův algoritmus, který v provedených testech dopadl nejlépe.

2.3.8 Existující distribuované souborové systémy

Dodnes bylo vytvořeno mnoho distribuovaných souborových systémů, některé z nich jsou stále využívány a dále rozvíjené, jiné zanikly. Podrobněji popsány budou pouze vybrané distribuované souborové systémy.

InterMezzo

Distribuovaný souborový systém, který vychází z projektu Coda[35]. Hlavním cílem je vytvoření distribuovaného souborového systému, který bude programově jednodušší, ale bude zachovávat výhody pokročilého protokolu jakým je Coda. Klíčovým rozhodnutím o designu bylo využití lokálního souborového systému jako úložného prostoru na serveru a jako klientské cache. InterMezzo[19] spoléhá na existující infrastrukturu, jako je například TCP[33], na rozdíl od AFS[16] nebo Cody, které vše implementovaly od základů v programovacím jazyce C.

Coda

Vývoj projektu Coda byl zahájen v roce 1987 na Carnegie Mellon University. Stejně jako AFS i tento vývoj vedl profesor M. Satyanarayanan. Předlohou pro Codu byl distribuovaný souborový systém AFS, konkrétně AFSv2. Oproti AFS nabízí několik nových prvků, které neobsahovaly žádné starší systémy. K hlavním vlastnostem patří:

- **Offline práce s daty** – pokud dojde k přerušení připojení mezi klientem a serverem, Coda dále používá data z lokální cache. Po znovu připojení nejprve dojde k zapsání dat na server. Pokud dojde ke kolizi, klient ji musí manuálně vyřešit. Následně klient pokračuje v normálním provozu.
- **Odolnost vůči chybám** – oproti AFS, které dovoluje pouze jednomu serveru zapisovat data, Coda umožňuje provádět opera čtení a zápisu na všech replikovaných serverech. Zavádí mechanismy pro řešení server/server konfliktů, obnovení po rozpojení sítě nebo znovu připojení klientů.
- **Výkon** – Coda je vysoce výkonný distribuovaný souborový systém, díky trvalé cache na straně klienta.
- **Bezpečnost** – autentizace je zajištěna protokolem Kerberos. Pro omezení přístupu k datům uživatelům je použit ACL.
- **Dobře definovaná sémantika sdílení**

Momentálně se systém stále nachází ve fázi vývoje, ale má velký potenciál stát se lepším distribuovaným souborovým systémem než AFS.

Ceph

Tento distribuovaný souborový systém byl vytvořen za účelem poskytnout excelentní výkon, spolehlivost a škálovatelnost. Ceph[43] maximalizuje oddělenost dat a správy metadat, tím že nahrazuje alokační tabulky pseudonáhodnou funkcí pro distribuované šíření dat jménem CRUSH[44]. Funkce CRUSH je navržena pro heterogenní a dynamické klastry nespolehlivých zařízení sloužící k uložení objektů (OSDs). Dynamicky distribuovaný metadatový klastr poskytuje efektivní zprávu metadat a širokou škálu využití pro jakékoliv účely. Ceph tímto řeší jeden z problémů, který se vyskytuje ve škálovatelných uložištích. Tento problém spočívá v tom jak efektivně zajistit

jednotnou hierarchii a zároveň zachovat vysoký výkon při velké škálovatelnosti. Testy ukázaly, že Ceph dosahuje excelentního výkonu v I/O operacích a ve správě metadat. Ceph dokáže provést až 250 000 operací za vteřinu.

GFS

Global File System[38] je distribuovaný souborový systém který je obsažen v Red Hat Enterprise Linuxu. Jedná se o clusterový souborový systém, to mu umožňuje čtení a zápis do jednoho sdíleného souborového systému. Pro sdílení úložných prostor používá Fibre Channel. Konzistence je zajištěna zamykáním souborů. Pro zamykání používá distribuované zámky.

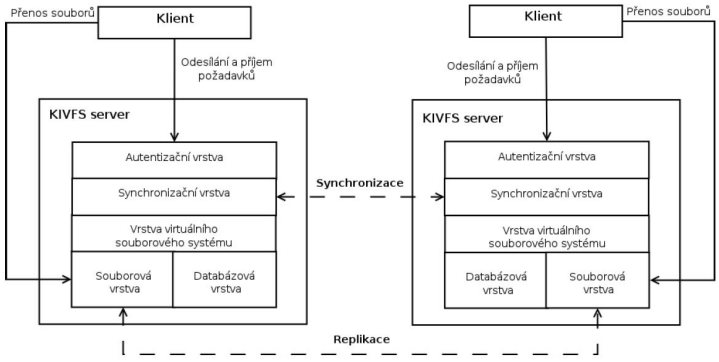
AFS

AFS (Andrew File System) byl vytvořen na Carnegie-Mellon University v roce 1982, pod vedením profesora M. Satyanarayanana. Jednolo se o vytvoření systému, který by byl použit pro velké počítačové sítě a byl schopný pracovat s tisícovkami uživatelů. Oproti jiným distribuovaným souborovým systémům má několik hlavních výhod v oblasti bezpečnosti a škálovatelnosti. V AFS existuje jeden hlavní uzel (master) a n uzlů vedlejších (slave). Používá replikace typu read-only a využívá model slabé konzistence. Nejpodstatnější na celém AFS je návrh jeho protokolu. Například v NFS[27] byl uživatel nucen periodicky kontrolovat, jestli obsah cache zůstal nezměněn. To mělo za následek velké využívání serverového výkonu. Protože v NFS[27] byla konzistence cache závislá na nízko úrovňové implementaci, bylo těžké ji popsat. Naproti tomu u AFS je to snadné. Kdykoliv totiž uživatel otevře soubor, vždy dostane poslední konzistentní verzi ze serveru. AFSv1 nebyla schopná pracovat v tak velkém měřítku, jak bylo zamýšleno a bylo nezbytné ji přepracovat. Ve finální verzi AFSv2 byly opraven problémy, které obsahovala AFSv1. Existuje několik implementací například OpenAFS[30] nebo Arla[2].

KIVFS

KIVFS je distribuovaný souborový systém vyvíjen na Katedře informatiky a výpočetní techniky, Fakulty aplikovaných věd na Západočeské univerzitě v Plzni. Hlavní motivací vzniku je fakt, že žádný z uvedených distribuovaných souborových systémů nesplňuje požadované vlastnosti jako jsou například read-write replikace nebo dynamické směrování požadavků. KIVFS je vyvíjen pro použití v heterogenním a nezabezpečeném prostředí. Je postaven na modelu klient – server. Komunikuje pomocí KIVFS protokolu, který je popsán v podkapitole 2.3.8. Jedná se o distribuovaný souborový systém,

které využívá transakční zpracování operací. Tato vlastnost je motivována potřebou mít na všech uzlech metadata ve stejném stavu. Model konzistence musí být tedy nejhůře sekvenčně konzistentní. To umožňuje používat replikace typu read-write, které zvyšují odolnost proti selhání. Každý uzel je schopný se po výpadku obnovit do aktuálního stavu. Aby bylo možné každý uzel správně obnovit, používá KIVFS logické hodiny pro určení pořadí požadavků. Lze přesně určit, v jakém stavu se uzel nacházel při výpadku a kolik operací proběhlo, než se uzel stal znovu aktivním. Jeho největší výhodou, oproti ostatním systémům, je dynamický routing. Tím je dosažen co možná největší výkon pro přenos dat, jak mezi klientem a serverem, tak mezi jednotlivými uzly.



Obrázek 2.5: Schéma KIVFS.[37]

Každý uzel je tvořen jako skupina vrstev, viz obr. 2.5. Každá vrstva má svojí specifickou funkci. Veškerá komunikace probíhá pomocí KIVFS protokolu, který je postaven nad TCP/IP. Spojení je vždy inicializováno směrem od klienta. Serverové části jsou naprogramovány v jazyce C, ale díky komunikace pomocí KIVFS protokolu, není implementace programovacím jazykem nijak omezena. Serverová část je určena k běhu na operačním systému GNU/Linux. Jednotlivé vrstvy jsou závislé na použitých knihovnách než na systému jako takovém. Každá vrstva plní zadanou funkčnost. Funkce vrstev jsou logicky uspřádané následovně:

- **Autentizační vrstva** - Tato vrstva má za úkol autentizaci uživatele a zabezpečenou komunikaci s klientskými aplikacemi. Pro zabezpečenou komunikaci používá protokol SSL[10]. Autentizace uživatele probíhá pomocí systému Kerberos. Funkčnosti autentizační vrstvy a problematice zabezpečení jsou věnovány bakalářské práce od Marka Pivničky[31] a Karla Vlčka[42].

- **Synchronizační vrstva** - Tato vrstva operuje napříč celou skupinou serverů KIVFS. Jejím úkolem je synchronizace požadavků na změnu metadat souborového systému i zpráv systému spravující repliky, zámky a další interní struktury. Podrobněji o replikacích a synchronizaci pojednává bakalářská práce od pana Michala Junáka[23] a diplomová práce od pana Jindřicha Skupy[37]. Řídí také obnovu uzlů. Dále má za úkol podle získaných informací hledat nejkratší cesty podle zvolených metrik, na základě, kterých následně jsou požadavky směrovány.

- **Vrstva virtuálního souborového systému** - Tato vrstva slouží jako prostředník pro rozdělování požadavků mezi podřízené vrstvy. Použití této vrstvy přidává do systému KIVFS nezávislost na změně databáze na databázovém serveru, změně struktury pro ukládání dat na souborovém serveru a nezávislost mezi databázovým a souborovým serverm. Návrh virtuálního souborového systému v rámci KIVFS je podrobně popsán v bakalářské[40] a diplomové práci[41] od pana Radka Strejce.

- **Databázová vrstva** - Hlavním úkolem je uchovávání metadat, neboli kompletní adresářovou strukturu, atributy jednotlivých souborů, informace o replikách, mapování virtuální adresářové struktury na skutečné soubory a jiné. Důvodem uchovávání metadat v databázi a nevytvářet fyzickou adresářovou strukturu, je úspora paměti, větší výkon a snadnější replikace celé databáze.

- **Souborová vrstva** - Řídí ukládání fyzických souborů na lokální systém souborů jednotlivých uzlů, poskytuje uživateli přímý přístup k datům, provádí replikace a řídí jejich přenos, šifrování a deduplikaci dat. Fyzické souboru, z důvodu jejich velikosti, není vhodné uchovávat v databázi. Docházelo by k navýšení celé velikosti databáze, což by vedlo ke snížení výkonu při replikacích. Muselo by se přenášet o mnoho větší množství dat, než při odděleném použití databázové a souborové vrstvy.

Spolu se serverem KIVFS bylo vytvořeno i několik klientů:

- **Webový klient** kterému je věnována bakalářská práce pana Karla Hovorky[15].

- **Klienta pro GNU/Linux s použitím jaderného modulu**. Na vývoj tohoto klienta je zaměřena bakalářská práce pana Jana Fazekaše[9].

- **Klient pro GNU/Linux s použitím FUSE**, kterému je věnována bakalářská[21] a diplomová[22] práce pana Přemysla Jaroše.

Systém KIVFS má i své problémy. Největším problém je fakt že na vývoji pracovalo několik týmů a jednotlivců. To způsobilo že kód je neudržovaný a nepřehledný. Dalším problémem je samotná architektura KIVFS. Protože tento systém je složen z vrstev, které na sobě fungují nezávisle, je komplikované systém nasadit a rozběhnout. Rozdělí do pěti nezávislých vrstev vyžaduje rozsáhlou komunikaci, aby byla dosažena výsledná funkčnost. To vede k velkému počtu zpráv, které je potřeba přenášet. K této komunikaci jsou využívány network sokety, což také není nejrychlejší možné řešení

KIVFS protokol Pro účely KIVFS byl vytvořen síťový protokol postavený nad TCP protokolem. Jde o binární protokol, který obsahuje hlavičku a balíček s daty, viz obr. 2.6. Tento protokol se používá pro komunikaci jak mezi klientem a serverem, tak mezi jednotlivými uzly navzájem. Hlavička obsahuje následující atributy:

- **Magické číslo** – toto číslo slouží k označení zprávy a je generováno náhodně.
- **Časová známka** – jedná se o unixový čas získaný při vytvoření hlavičky.
- **Kód požadavku** – číselný kód zvoleného požadavku.
- **Návratový kód** – identifikace, jestli vše proběhlo správně. Jako i v jiných systémech, tak i tady znamená 0, že je vše v pořádku.
  V opačném případě je návratový kód roven kódu označující danou chybu.
- **Velikost balíčku s daty** – velikost balíku s daty. Data obsahují následující atributy:
- **Velikost záznamu** – velikost záznamu, který obsahuje data.
- **Záznam** – jedná se o textový řetězec přenášených dat.

Zpráva musí vždy obsahovat hlavičku. Není ale vždy nezbytné, aby obsahovala balíček s daty. Například, pokud slouží hlavička pouze jako odpověď, jestli byl daný příkaz správně vykonán, bude ve velikosti balíčku s daty uvedená velikost 0. Příjemce po přečtení velikosti 0 ví, že balíček s daty není ve zprávě obsažen.

Obecně	
Hlavička	
Magické číslo	123456
Časová známka	1490030521
Kód požadavku	3
Návratový kód	0
Velikost dat	10
Data	
Velikost záznamu	6
Záznam	/kivfs

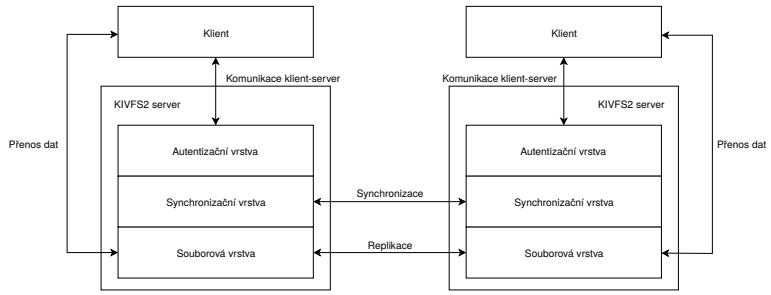
Výpis adresáře
Hlavička
123456
1490030521
3
0
10
Data
6
/kivfs

Obrázek 2.6: Zpráva KIVFS.

2.3.9 Návrh KIVFS2

Systém KIVFS2 vychází ze systému KIVFS.

Původní systém KIVFS obsahoval pět vrstev, viz obr. 2.5. Nově navržený systém má pouze tři vrstvy, viz obr. 2.7. Byla odstraněna vrstva Virtuálního souborového systému a Databázová vrstva byla přesunuta do Synchronizační vrstvy. Touto úpravou bylo docíleno úspory komunikace.



Obrázek 2.7: Architektura KIVFS2.

Do systém KIVFS2 byla nově zavedena možnost pro komunikaci mezi jednotlivými vrstvami používat i unixové sokety, na rozdíl od starého systému KIVFS, který používal jen network sokety. Na základě testování, jehož výsledky jsou uvedeny v tab. 2.1, bylo zjištěno, že při použití unix soketů dojde ke zvýšení rychlosti, v komunikaci mezi vrstvami, o **+38,2%**.

Měření	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	Průměr
Čas s Network soket (μs)	335	346	371	340	362	351	342	369	362	352	353
Čas s Unix soket (μs)	206	203	229	238	207	220	205	230	208	235	218,1

Tabulka 2.1: Výsledky testování s použitím network soketu a unix soketu.

Architektura KIVFS2

KIVFS2 je založen na síťové architektuře klient – server. Veškerá komunikace probíhá pomocí KIVFS protokolu, viz. kapitola 2.3.8, který je postaven nad TCP/IP. Každý server se skládá ze tří vrstev. Jedná se o autentizační vrstvu, synchronizační vrstvu a datovou vrstvu. Každá vrstva je spuštěna ve vlákně. Mezi sebou vrstvy mohou komunikovat přes síťové sokety nebo přes unixové sokety. Obě varianty lze použít také v šifrované podobě. Pro zvýšení efektivity, KIVFS2 pracuje s metadaty, která jsou uložena v databázi. Fyzické soubory nahrané uživatelem jsou uchovávány na pevném disku. Na obr. 2.7 je vyobrazená architektrua KIVFS2.

Autentizační vrstva – Přes tuto vrstvu procházejí všechny požadavky od klienta. Má za úkol autentizaci a autorizaci uživatele. Autentizace probíhá skrz systém Kerberos, a je vyžadována při prvním připojení na server. Jakmile je uživatel ověřen, všechny jeho požadavky jsou předávány synchronizační vrstvě.

Synchronizační vrstva – Tato vrstva má za úkol synchronizovat veškerý provoz mezi jednotlivými uzly KIVFS. Stará se tedy o synchronizaci logických hodin, synchronizaci požadavků a zpráv, používaných napříč celým systémem. Dále zajišťuje kompletní správu metadat uložených v MySQL databázi a stará se o obnovu po výpadku uzlu a směrování požadavků. Aby mohl správě směrovat tok dat a požadavků, udržuje si informace o ostatních uzlech a hledá nejkratší cesty. Komunikuje s ostatními uzly napříč systémem a také s datovou vrstvou.

Souborová vrstva – Nejníže položená vrstva slouží jako datové uložiště pro všechny fyzické soubory ležící na daném uzlu. Zajišťuje správu fyzických souborů uchovávaných na serveru. Dále tato vrstva zajišťuje veškerou správu replikací a přenos souborů mezi serverem a klientem.

Cíle ověření

Na systému KIVFS2 budou provedeny stejné testy jako na původním systému KIVFS. Konkrétně se jedná o tyto výkonnostní test:

- Upload souboru bez použití dynamického routingu
- Upload souboru s použitím dynamického routingu

- Download souboru bez použití dynamického routingu
- Download souboru s použitím dynamického routingu
- Upload 100 malých souborů

Všechny tyto testy budou provedeny se soubory o velikosti 10MB, 100MB a 1GB. Linky použité pro testování budou mít rychlost 10Mbps, 100Mbps a 1000Mbps. Podrobněji jsou testy popsány v kapitole 4.

Získané výsledky, z provedeného testování, budou porovnány s výsledky původních testů systému KIVFS. Toto porovnání by mělo dopadnou lépe pro systém KIVFS2.

3 Praktická část

Pro implementaci byl zvolen programovací jazyk C. Jazyk C byl vybrán, protože se jedná o systémový jazyk operačního systému Linux, dává programátorovi větší kontrolu nad programem a výsledný program bývá rychlejší a zabírá méně paměti než programy napsané ve vysokoúrovňových jazycích. K přeložení programu je použit překladač gcc a k sestavení se používá GNU make[39].

3.1 Síťová komunikace v KIVFS2

Pro síťovou komunikaci v KIVFS2 je použit protokol KIVFS, který je popsán v kapitole 2.3.8. Jedná se o binární protokol realizovaný nad TCP. Z důvodu optimalizace, byla zvažována možnost použití UDP[32] protokolu, nakonec byl vybrán protokol TCP. Použití TCP protokolu místo UDP je z důvodu zachování správného pořadí zpráv a eliminaci případné ztráty, bez nutnosti další implementace.

3.2 KIVFS2: Autentizační vrstva

Autentizační vrstva je jediným vstupem do systému KIVFS2. Jejím hlavním úkolem je spravovat přístupy do systému a zabezpečení. Tato vrstva provádí autentizaci a autorizaci uživatele. Klient se musí úspěšně ověřit, než je mu umožněna další komunikace. Dále tato vrstva funguje jako mezičlánek mezi klientem a synchronizačním serverem. Protože tato vrstva se chová jako proxy, prochází skrz ni i vzájemná komunikace dvou uzlů. Na rozdíl od klienta, se jiný uzel musí pouze autorizovat. Pro zabezpečenou komunikaci s klientem používá protol SSL.

3.2.1 Autentizace

Pokud chce klient komunikovat se serverem, je nezbytné, aby prokázal svou totožnost. Po navázání spojení je zahájen proces autentizace. Pokud je autentizace úspěšná, klient je puštěn dále k procesu autorizace. Ověření probíhá na základě ověření uživatelského jména a hesla. V případě, že klient neprokáže úspěšně svoji totožnost, dojde k jeho odpojení.

Samotná autentizace je realizovaná pomocí síťového autentizačního protokolu Kerberos. Funkčnost Kerbera byla popsána v kapitole 2.3.1. Jako implementace Kerbera je v KIVFS2 použit Heimdal. Heimdal byl zvolen na základě provedeného testování v bakalářské práci od pana Marka Pivničky[31]. Jedná se o implementaci, která vznikla ve Švédsku a je volně dostupná pod BSD licencí. Heimdal nabízí několik knihoven pro různé programovací jazyky. Dále nabízí širokou škálu šifrovacích algoritmů. Šifrování autentizace je obsaženo v samotném protokolu Kerberos.

3.2.2 Autorizace

Po úspěšném ověření totožnosti, musí klient získat povolení k přístupu do samotného distribuovaného souborového systému. Autorizace se provádí ověřením uživatelského jména proti lokální databázi uživatelů. Popis databáze v KIVFS2 je uveden v podkapitole 3.3.6. Jakmile je uživatel autorizován, je mu umožněna další komunikace se serverem a jeho požadavek je předán synchronizační vrstvě. V opačném případě je mu sdělena chybová hláška a dojde k jeho odpojení.

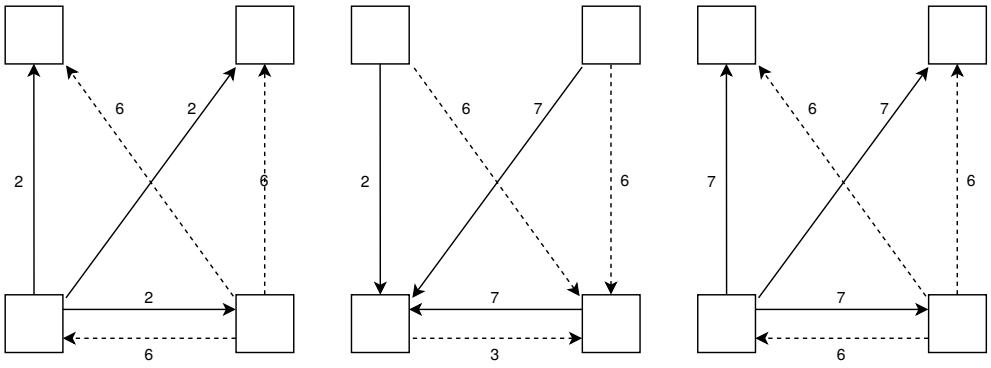
3.3 KIVFS2: Synchronizační vrstva

Úkolem synchronizační vrstvy je synchronizovat veškerý provoz v KIVFS2. Přijímá požadavky, které předává autentizační vrstva a dále s nimi pracuje. Jedním z hlavních úkolů je zajistit správně pořadí jednotlivých požadavků a jejich transakční provedení. K tomu to účelu se v KIVFS2 používají logické hodiny. Dalším úkolem je obnovení uzlu po výpadku. Při spuštění je vždy porovnán aktuální čas získaný od ostatních uzlů proti poslednímu času, který byl zapsán do databáze. Pokud se liší, přistoupí se k obnovení. Uzel nepřijímá žádné zprávy, dokud není vše aktuální. Aby mohlo KIVFS2 používat dynamický routing, je nezbytné hledat nejrychlejší cesty z daného uzlu do všech ostatních. Každý uzel si uchovává informace o ostatních a u každého uzlu si uchovává nejrychlejší cestu do daného uzlu, přes kterou probíhá komunikace. Hledání nejrychlejších cest je realizováno v periodických intervalech. Interval je možné nastavit v konfiguračním souboru serveru.
K hledání nejrychlejších cest je použit Dijsktrův algoritmus. Ten byl vybrán na základě provedeného porovnání v diplomové práci pana Jindřicha Skupy[37]. Tato vrstva také komunikuje přímo s MySQL databází, které je klíčová pro většinu funkčnosti v KIVFS2. Důvodem pro použitý databázového systému MySQL je jeho rychlost při práci s daty.

3.3.1 Synchronizace požadavků

Pro synchronizaci požadavků je použito skalárních hodin. Každý uzel systému KIVFS2 čísluje všechny operace, které provádí. K synchronizaci dochází při komunikaci s jiným uzlem, vždy je vybráno vyšší číslo. Algoritmus logických skalárních hodin popsal Leslie Lamport. Každý uzel má své logické hodiny, podle kterých se každé operaci přiřadí číslo.

Tento algoritmus má tu nevýhodu, že nedokáže řešit konkurenční požadavky, což je pro KIVFS2 nezbytné. Z tohoto důvodu se zavádí drobná úprava tohoto algoritmu. Uzel, který přijal požadavek od klienta, se stává koordinátorem a postupuje následovně. Nejprve hodnotu svých hodin přiřadí operaci, kterou chce provést. Následně odešle zprávu ostatním uzlům. Tato zpráva obsahuje operaci a hodnotu, pod kterou se má provést. Každý uzel po přijetí porovná hodnotu svých logických hodin a hodnotu obsaženou ve zprávě. Následně přiřadí vyšší hodnotu z porovnaných do zprávy
s odpovědí, kterou odešle zpět a podle potřeby aktualizuje své lokální hodiny. Koordinátor přijímá odpovědi a porovnává jejich hodnoty se svými lokálními hodinami. Pokud je hodnota ve zprávě vyšší aktualizuje svoje hodiny. V posledním kroku všem ostatním uzlům odesílá hodnotu, pod kterou se má operace provést. Operace je uložena do fronty pod domluvenou hodnotou logických hodin a uzel čeká, až ji bude moci provést. Příklad použití tohoto algoritmu je ukázán na obr. 3.1.



Obrázek 3.1: Algoritmus použitý pro synchronizaci požadavků.

3.3.2 Dynamické směrování požadavku

Aby bylo možné dynamicky směrovat požadavky po nejrychlejších cestách, je nejprve nezbytné tyto cesty nalézt. K hledání nejrychlejších cest je používán Dijkstrův algoritmus. Každé lince je přiřazena hodnota podle metriky. Pro KIVFS2 byl zvolen stejný výpočet metriky jako v KIVFS. Tento výpočet

popsal ve své diplomové práci pan Jindřich Skupa[37]. Na základě této hodnoty probíhá vyhledávání nejrychlejších cest. Nalezené cesty jsou ukládány do tabulky na jednotlivých uzlech. Každý uzel si udržuje informace o všech ostatních uzlech. Mezi těmito informacemi je i nejrychlejší cesta z daného uzlu do uzlu cílového. Pokud je při komunikaci zjištěno že existuje rychlejší cesta než ta přímá, je po ní požadavek směrován. Protože může docházet ke změnám na fyzických linkách, je nezbytné provádět vyhledávání nejkratších cest v periodických intervalech. KIVFS2 vyhledává nejkratší cesty každé tři hodiny. Pokud by byl interval příliš krátký, mohlo by docházet k velkému zatížení systému.

3.3.3 Tunelování požadavků

V systému KIVFS2 je využíváno tunelovaného spojení mezi uzly. Tunelování je propojení dvou uzlů skrz jiné uzly, za účelem dosažení co nejrychlejšího možného přenosu dat. K tunelování požadavků je využito dynamického směrování. Aby bylo možné tunelování využívat, musí být nejprve nalezeny nejrychlejší cesty mezi jednotlivými uzly. Při odesílání požadavku se systém nejprve podívá, jestli existuje rychlejší cesta než ta přímá, pokud ano požadavek pošle po ní. Pokud cesta vede přes jeden nebo více uzlů, jsou na této cestě vytvořeny tunely. První uzel na cestě je informován o vytváření tunelovaného spojení a vytvoří si příslušné vlákno. Toto vlákno dále přijímá zprávy a předává je na další uzel, který je také tunel nebo přímo na cílový uzel. Cílový uzel následně zprávu přijme, zpracuje ji a odpověď posílá znovu skrz tunel zpět.

3.3.4 Vyloučení souběžného přístupu

Aby bylo zabráněno dvěma uzlům v nahrávání toho samého souboru současně, systém KIVFS2 používá systém zámků. Tyto zámky jsou realizované na metadatové úrovni. Při uploadu souboru je soubor v databázi zamčen na všech uzlech, kde je replikován. Toto zamykání je realizováno v tabulce *Status*. Soubor je odemčen po úspěšném nahrání na daný uzel. Pokud uživatel nahrává soubor, který se dále replikuje, je tento soubor odemčen po úspěšném nahrání a replikace probíhají na pozadí. Uzel odemkne svou replikaci po jejím úspěšném přijetí.

3.3.5 Ukládání metadat

Systém KIVFS2 používá pro uchovávání informací o adresářové struktuře metadata. Data jsou uložena v databázi a distribuována na všechny uzly.

Použití metadat má několik výhod. Ta první je, že metadata nejsou nijak náročná na paměť, což umožňuje mít vždy aktuální data na všech uzlech. Další výhodou je rychlost, která je vyšší než při použití fyzické adresářové struktury. V databázi jsou uloženy informace o mapování souborů, které jsou fyzicky uloženy na souborovém systému serveru.

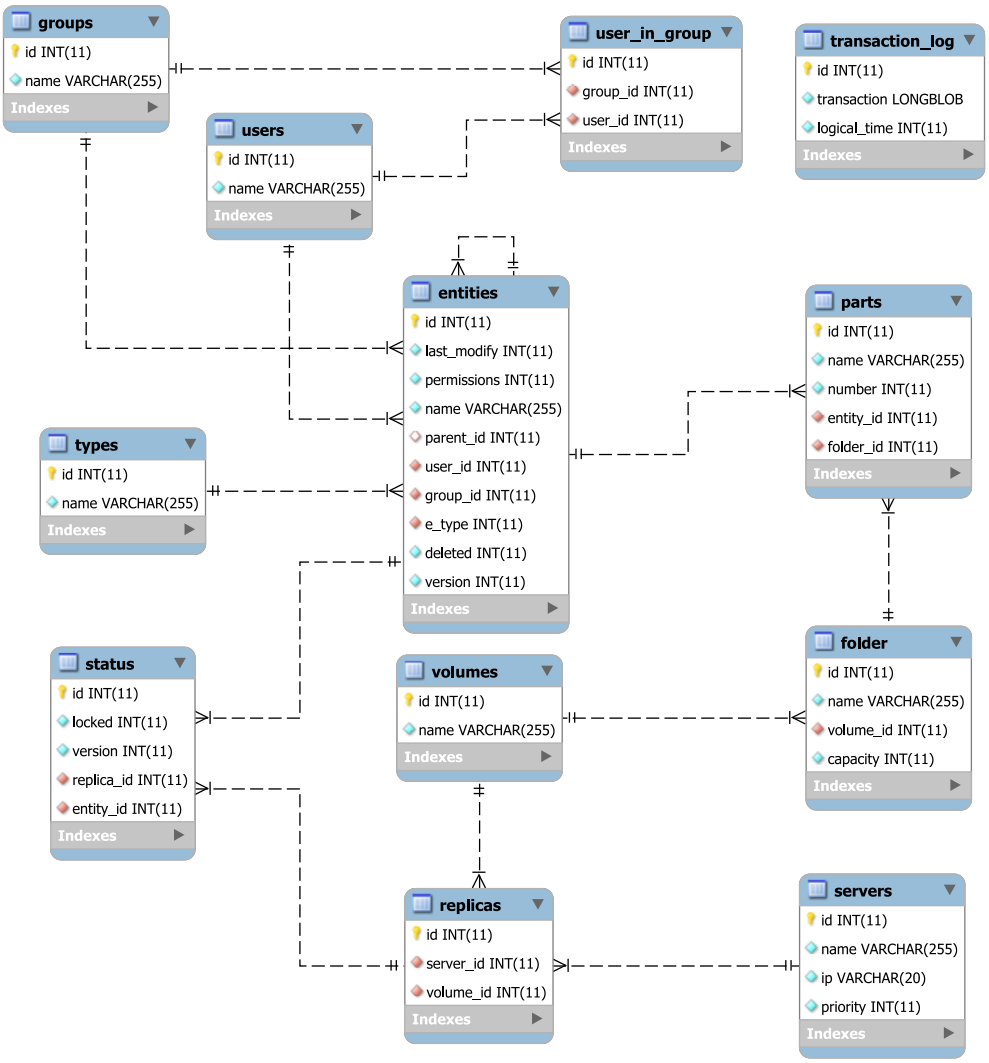
3.3.6 Databáze v KIVFS2

V KIVFS2 je použita MySQL databáze a je znázorněna ERA modelem na obr. 3.2. Databáze byla navržena tak, aby splňovala všechny požadované vlastnosti, které na ni byly kladeny. Tyto vlastnosti jsou:

- Udržování informací o uživatelích KIVFS2
- Udržování informací o všech entitách
- Udržování informací o replikacích
- Umožňovat rozdělení souboru na části
- Musí být možné zamykat soubor, pokud je uploadován nebo replikován
- Udržování transakčního logu pro obnovení uzlu

Výsledný ERA model splňuje všechny tyto vlastnosti. Z databáze lze tedy získat například, kam který server má replikovat, kde fyzicky leží části některé z entit, která z entit je zamčená atd. Popis jednotlivých tabulek:

- **Tabulka Groups** – Uchovává uživatelské skupiny, aby bylo možné uživatelům přidělovat přístup na základě toho, v jaké skupině se nachází.
- **Tabulka Users** – Uchovává id a jméno uživatele, kteří mají přístup do KIVFS2.
- **Tabulka User_in_group** – Rozložení vztahu *M:N* mezi tabulkami *Groups* a *Users*.
- **Tabulka Type** – Uchovává typ entity (momentálně soubor nebo složka).
- **Tabulka Entities** – Nejdůležitější tabulka, které uchovává informace o všech entitách existujících v KIVFS. Tato tabulka udržuje informace jako je jméno entity, unikátní id, id vlastníka, id skupiny, typ entity, kdy byla naposledy upravena, oprávnění, atribut jestli byla entita smazána, verzi entity a jako poslední je id rodiče. Pokud rodič neexistuje, je atribut *NULL*.



Obrázek 3.2: Model databáze KIVFS2.

- **Tabulka Servers** – Uchovává informace o všech serverech v KIVFS2.
- **Tabulka Replicas** – Uchovává informace o replikacích. Na základě této tabulky server ví, na jaký svazek má replikovat.
- **Tabulka Volumes** – Uchovává informace o všech svazcích v KIVFS2.
- **Tabulka Folder** – Uchovává informace o všech složkách v KIVFS2 a na kterém svazku se nacházejí.
- **Tabulka Status** – Tato tabulka slouží k zamykání entit na určitém serveru.
- **Tabulka Parts** – Uchovává informace o částech některé z entit, která je typu soubor.

3.3.7 Databázový log

Poslední tabulkou je tabulka *Transaction_log*. Tato tabulka slouží jako transakční log a nemá žádnou vazbu na ostatní tabulky. Uchovává operaci, která byla nad KIVFS2 databází provedena a čas, kdy byla provedena. Pomocí této tabulky je prováděna obnova uzlu po výpadku. Díky logickým hodinám lze přesně určit od jaké transakce se má obnova provádět.

3.3.8 Mazání dat

Jak již bylo řečeno, KIVFS2 ukládá informace ve formě metadat. Pokud uživatel chce smazat nějaká data, KIVFS2 nastaví atribut *deleted* na hodnotu 1, která říká, že soubor nebo složka je smazaný a například při výpisu adresáře se nevypíše. Momentálně k samotnému procesu smazání, dochází ihned, ale lze nemusí tomu tak být. Systém mazání, kdy jsou metadata značena jako smazána má několik výhod. Lze tento systém využít tak, že data nebudou smazána ihned, ale například v nočních hodinách, aby nedocházelo ke snížení výkonnosti systému, za pomocí Cronu. Další z nich je, že pokud si to uživatel rozmyslí, je možné data snadno obnovit. Pokud je smazán soubor, dojde i k odstranění jeho fyzického souboru na souborovém systému.

3.3.9 Replikace databáze

Data, se kterými pracuje KIVFS2 se dělí na dvě části. Metadatová část uložená v databázi a fyzické soubory uložené na souborovém systému. Protože metadata nejsou paměťově náročná jako fyzické soubory, mohou být

replikována na každý uzel v systému. KIVFS2 používá read-write replikace, každý uzel může provádět změnu. Replikace v systému KIVFS2 musí být alespoň sekvenčně konzistentní, aby bylo možné považovat všechny repliky za rovnocenné. Změny provedená v databázi jsou šířeny aktivně při provádění transakce. Každý uzel dostane informaci o tom, jaká operace se má provést a mění si svou repliku sám, tím že provádí operace ve stejném pořadí jako ostatní uzly. Uzly si mezi sebou předávají pouze informace o tom, které operace mají provádět.

3.3.10 Zotavení po chybě

Pokud dojde k havárii některého z uzlů, je tento uzel schopný se po znovuspuštění obnovit do aktuálního stavu. Při spuštění uzel zažádá ostatní uzly o jejich stav logických hodin. Vybere ten nejvyšší a ten porovná s hodnotou posledního provedeného požadavku, který byl zapsán do transakčního logu. Pokud zjistí, že aktuální hodnota je vyšší, zahájí proces obnovy. Tento proces lze rozdělit do dvou částí. První je obnova metadatové části. Ta probíhá tak, že obnovovaný uzel zažádá jiný uzel, (protože je v systému použito aktivní replikování je jedno jaký uzel požádá) aby mu poslal všechny provedené operace od jeho posledního zapsaného času. Jakmile přijme odpověď obsahující provedené operace, zahájí jejich zpracování. Jelikož během obnovy uzel nekomunikuje s ostatními, může dojít k situaci, že při obnově byly provedeny další operace. Opakuje tedy proces žádání a zpracování odpovědi až do chvíle kdy je jeho čas aktuální. Druhou částí je obnovení fyzických souborů, které měl uzel přijmout během doby, kdy nebyl v provozu. Tato obnova je popsána v podkapitole 3.4.3.

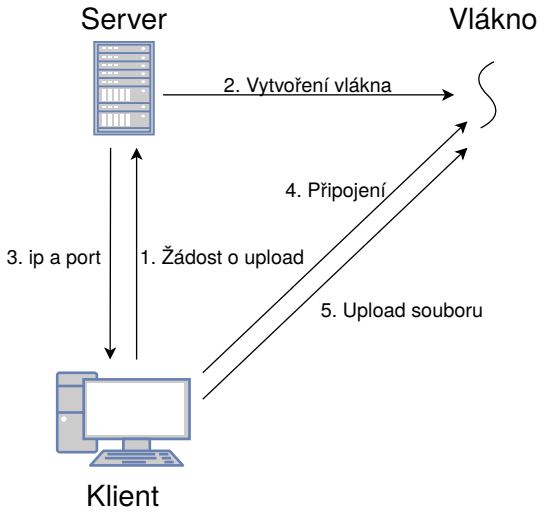
3.4 KIVFS2: Souborová vrstva

Jedná se o nejníže položenou vrstvu. Jejím hlavním úkolem je veškerá správa souborů uložených na fyzickém disku a jejich replikace. Tato vrstva je schopna komunikovat přímo s klientem při nahrávání nebo stahování souborů a s ostatními uzly při replikaci nebo poskytování souborů. Řídí se metadaty. Ta určují, kam se daný soubor nebo jeho část uloží a pod jakým jménem. Jméno je id souboru, pod kterým je uložen v databázi. Metadata dále určují, na jaký server se mají replikovat data, která jsou nahrávána na určitý uzel. Přenos dat je možné směrovat stejným způsobem jako tomu je u směrování požadavku. Směrování na úrovni souborové vrstvy využívá stejnou tabulku a také používá systém tunelů.

3.4.1 Princip uložení souborů

V systému KIVFS2 jsou fyzické soubory rozděleny a ukládány po částech na lokální souborový systém. Dočasné soubory, které ještě nebyli úspěšně nahrány, jsou ukládány do dočasného uložiště */mnt/kivfs2/tmp*. Po úspěšném přenosu jsou přesunuty na svoje cílové uložiště ve tvaru */mnt/kivfs2/jméno svazku/jméno složky/*. Jméno svazku a jméno složky je získáno z databáze metadat. Ukládání po částech probíhá při nahrávání dat. Jakmile má jedna část požadovanou velikost, je uzavřena a vytvořena další část. Informace o jednotlivých částech jsou uchovávány v databázi metadat. Tento přístup k ukládání, přináší velkou výhodu pro replikace. Místo replikace celého souboru, lze provádět replikaci jen jeho částí. Soubor je ukládán pod názvem ve tvaru *id_část*, kde id je id daného souboru z tabulky *Entities*. Tedy například když id bude 5 a bude se jednat o první část, zapíše se na disk pod názvem *5_0*. Soubor, respektive jeho části, jsou ukládány do svazků. Cesta, kde má být server uložen je nejprve vytvořena v databázi. Je také možné omezit počet souborů v jedné složce a při dosažení maximálního počtu začít ukládat části do složky jiné.

3.4.2 Upload souborů

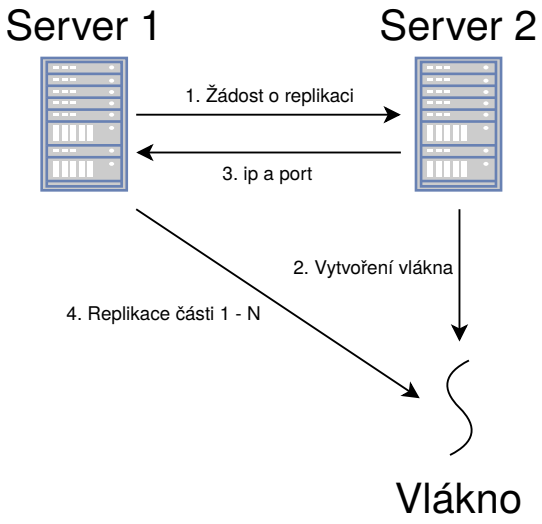


Obrázek 3.3: Upload souboru.

Upload souborů v systému KIVFS2 je realizován podobným způsobem jako přenos dat v pasivním režimu FTP[34]. Klient nejprve pošle žádost o upload souboru. Server tuto žádost zpracuje, a pokud soubor není uvedený v metadatech, tak jej vytvoří. Dále se id souboru předá souborové vrstvě.

Pod tímto id se soubor ukládá, viz podkapitola 3.4.1. Současně se synchronizační vrstva uzamkne. Souborová vrstva si připraví vlákno, na kterém bude přijímat soubor, který chce uživatel nahrát. Po tom, co je vlákno vytvořeno, odesílá se zpět uživateli odpověď, která obsahuje ip a port, na který se má připojit. Ihned po připojení zahájí uživatel přenos souboru, který je po částech ukládán na disk. Po ukončení přenosu je soubor odemčen. Upload souboru je ukázán na obr. 3.3.

3.4.3 Replikace souborů



Obrázek 3.4: Replikace souboru.

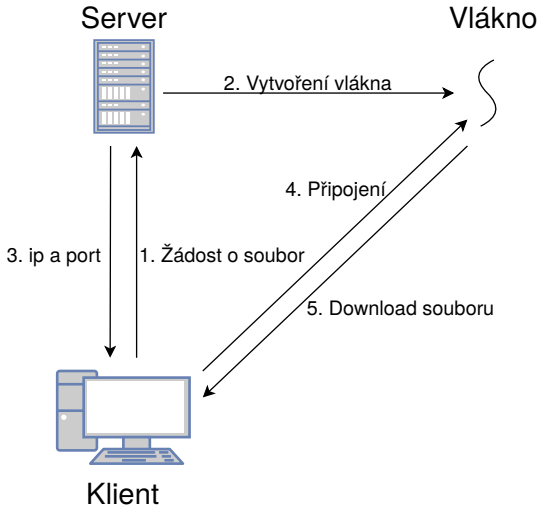
Jak již bylo řečeno, KIVFS2 používá read-write replikace. To znamená, že každý uzel může zapisovat. Replikace probíhá během uploadu. Kam má, jaký uzel replikovat určuje administrátor a informace o tom kam se replikuje je uložena v databázi. Po nahrání a uzavření první části souboru je spuštěno nové vlákno pro replikaci. Pokud uzel nemá kam replikovat, toto vlákno se nepustí. V opačném případě spuštěné vlákno kontaktuje ostatní servery, na které má replikovat, a zahájí paralelní přenos již úspěšně nahraných částí souboru. Po přijetí všech částí je daná replika odemčena a je dostupná uživateli. Replikace probíhají asynchronně na pozadí. Není tedy třeba čekat s odemčením na dokončení všech replikací.

V databázi metadat je zaznamenána verze každé repliky, po nahrání nové verze je tato hodnota zvýšena. Může nastat situace, kdy uzel zjistí, že mu některá replikace chybí. K této detekci dochází při obnovení po výpadku na základě tabulky *Status*. Pokud uzel zjistí, že v tabulce status existuje zamčená replika, zahájí obnovu. Kontaktuje některý ze serverů, na kterém

je nejvyšší verze požadovaného souboru a soubor si od něj nechá poslat. Po úspěšném přenesení odemkne svou repliku. Princip replikace souboru je ukázán na obr. 3.4

Systém replikací podporuje použití tunelového spojení. Jedná se o volitelnou možnost, která umožňuje přenášet replikace na základě dynamického směrování, viz. kapitola 3.3.1. Momentálně KIVFS2 provádí dynamické směrování pouze na základě rychlosti a odezvy linky. Je ale možné dále přidávat parametry, podle kterých má dynamické směrování probíhat. Například, pokud by existovali dvě linky, které by byli výkonnostně stejné ale lišila by se jejich cena.

3.4.4 Download souborů



Obrázek 3.5: Download souboru.

Uživatel požádá server o stažení souboru. Následně je vyhledáno id, pod kterým je uložen v databázi. Toto id se předá souborové vrstvě, která si připraví vlákno pro přenos souboru. Po připojení uživatele je požadovaný soubor odesílán, po částech, uživateli. Download souboru je ukázán na obr. 3.5. Může nastat případ, kdy požadovaný soubor neleží na daném serveru. Řešení je realizováno dvojím způsobem. První způsob je takový, že server zjistí, kde soubor skutečně leží, a kontaktuje příslušný uzel. Ten mu v odpovědi pošle ip a port, kde je pro něj připraven požadovaný soubor. Tyto údaje se předají uživateli, který si požadovaný soubor stáhne přímo z jiného koncového uzlu. Druhý způsob je takový, že server opět zjistí, kde leží požadovaný soubor. Kontaktuje tento soubor, ale odpověď s ip a portem již nepředá uživateli, ale spustí nové vlákno, které bude fungovat jako tunel mezi uživa-

telem a jiným uzlem. Tato varianta je realizována z toho důvodu, aby bylo dosaženo co nejrychlejšího přenosu. Pro klienta se nic nemění, protože vždy obdrží ip a port, kam se má připojit.

3.5 Přeložení a spuštění

Zdrojové kódy jsou organizovány v samostatných adresářích a jsou k dispozici na přiloženém CD v adresáři *KIVFS2*. Celá struktura vypadá takto:

```
KIVFS2
├── Core
├── Server
│   ├── Wrapper
│   ├── Sync
│   └── Fs
├── Client
├── Sql
└── MAKEFILE
```

Program využívá dvě externí knihovny, jedná se o knihovnu *ini.c*[18] a knihovnu *dijkstr.c*[7]. Překlad systému KIVFS2 se provádí programem *make*. Program make se spouští z hlavního adresáře *KIVFS2*, po jeho provedení jsou vytvořen dva spustitelné programy, jedná se o*kivfs2_server* a *kivfs2_client*.

Kerberos

Pro autentizaci byl zvolen protokol Kerberos konkrétně jeho implementace Heimdal. Každý účet Kerbera se nachází v logické entitě nazývané REALM. Pro KIVFS2 byl zvolen stejný název, jako pro KIVFS, *DFS.ZCU.CZ*. V tomto REALMU musí mít každý uživatel nebo služba vytvořený principal, což je identita uživatele. KIVFS2 využívá knihovnu *krb5*.

Instalace a nastavení serveru Pro správné fungování autentizační služby v systému KIVFS2 je zapotřebí správně nakonfigurovat Kerberos server. K tomu bude potřeba získat následující balíčky:

- heimdal-servers
- heimdal-servers-x
- heimdal-kdc

Tyto balíčky se instalují příkazem

apt-get install heimdal-servers heimdal-servers-x heimdal-kdc

Konfigurace serveru Kerbera se provádí modifikací soubor */etc/krb5.conf*. Ukázka konfiguračního souboru, viz obr. 5.1. Nastavení KDC je realizováno správou souboru */etc/heimdal-kdc/kdc.conf*.

Po dokončení základní konfigurace je možné přistoupit k nastavení serveru. Nejprve je nezbytné vytvořit hlavní klíč, kterým je chráněna veškerá práce, příkazem

kstash

Ke správě KDC je použit řádkový klient, který se spouští příkazem

kadmin -l

Po přihlášení je potřeba inicializovat databázi do které se ukládají principaly. K inicializaci slouží příkaz

kinit DFS.ZCU.CZ

První principal, který bude vytvořen, bude administrátorský principal a to příkazem

add administrator/admin@DFS.ZCU.CZ

Standartní uživatel se přidává příkazem

add "jméno uživatele"@DFS.ZCU.CZ

Další principal, který je nezbytný pro správné fungování, je principal pro službu. K přidání služby slouží tento příkaz

add -r "jméno služby"@"DNS serveru"

Uložený principal bude uložen do souboru příkazem

ext_keytab -k kivfs2.keytab "jméno služby"@"DNS serveru"

Tento vygenerovaný keytab bude umístěn na server, kde běží daná služba. KDC se spouští příkazem

/etc/init.d/heimdal-kdc start

Instalace a nastavení klienta Stejně jako u serveru, potřebuje klient následující balíčky

- heimdal-clients
- heimdal-clients-x
- heimdal-dev

Tyto balíčky se nainstalují příkazem

apt-get install heimdal-clients heimdal-clients-x heimdal-dev

Konfigurace klienta Kerbera se provádí modifikací soubor */etc/krb5.conf*. Ukázka konfiguračního souboru, viz obr. 3.6.

```
[libdefaults]
    default_realm = DFS.ZCU.CZ


[realms]
    DFS.ZCU.CZ= {
        kdc = ul403ds10-kiv.fav.zcu.cz
        admin_server = ul403ds10-kiv.fav.zcu.cz
    }

[domain_realm]
    .fav.zcu.cz = DFS.ZCU.CZ
    fav.zcu.cz = DFS.ZCU.CZ
```

Obrázek 3.6: Ukázka konfiguračního souboru Kerberos klienta.

Spuštění serveru KIVFS2

Než je možné spustit server je nezbytné ve složce */mnt* vytvořit adresář s názvem *kivfs2*. Jedná se o adresář, ve kterém jsou uchovávána všechna fyzická data systému KIVFS2 a server počítá s jeho existencí.

mkdir /mnt/KIVFS

Dále je nezbytné vytvořit databázi, ve které server uchovává všechna metadata. Pro vytvoření se nachází v podadresáři *Sql* sql skript s názvem

db.sql.

Sql skript se spouští příkazem *mysql -u "username"-p < db.sql*

Tento skript obsahuje základní informace o všech uzlech, replikacích atd. Změny týkající se nastavení například ip adres jednotlivých serverů je nutné měnit v tomto skriptu.

KIVFS2 server se spouští příkazem *./kivfs2_server config_s.ini*

config-s.ini je konfigurační soubor, viz. obr. 5.2, obsahující základní nastavení serveru. Definuje, na jakém portu poslouchá, na jakém portu běží ostatní vrstvy, jestli používá zabezpečenou komunikaci atd. Vysvětlení jednotlivých položek konfiguračního souboru:

- *net_ip* – ip adresa, na které server přijímá požadavky od klienta
- *net_port* – port, na kterém server přijímá požadavky od klienta
- *net_encoding* – šifrování komunikace s klientem. 0 – bez šifrování, 1 – šifrování
- *severity_level* – úroveň logování
- *ip* – ip adresa pro komunikaci s dalšími vrstvami
- *port* – pro komunikaci s dalšími vrstvami
- *encoding* – šifrování komunikace s ostatními vrstvami
- *pathname* – jméno unix soketu
- *node_port* – port na kterém, synchronizační vrstva přijímá systémové požadavky
- *node_encoding* – šifrování komunikace pro systémové požadavky
- *id* – id uzlu
- *username* – jméno uživatele pro přístup do databáze
- *password* – heslo uživatele pro přístup do databáze
- *database* – název databáze
- *hostname* - kerberos hostname

- *sname* - kerberos sname
- *kt_pathname* - kerberos cesta k souboru keytab
- *tunnel* - povolení tunelového spojení. 0 - nepoužívat, 1 - používat

Spuštění klienta KIVFS2

V rámci implementace server byl vytvořen klient, který je určen pro testování funkčnosti. Jedná se o řádkového klienta, viz. obr. 3.7, který má implementované základní požadavky, uvedené v tab. 3.1.

KIVFS2 client se spouští příkazem *./kivfs2_client config_c.ini*

config-c.ini je konfigurační soubor obsahující základní nastavení klienta. Definuje jeden až *n* serverů, ke kterým se klient bude připojovat. U každého serveru musí být uvedeno, jestli používá zabezpečenou komunikaci nebo nikoliv.

Obrázek 3.7: Klient - Ukázka výpisu adresáře.

Požadavek	Popis požadavku
mkdir "název_adresáře"	Vytvoří adresář s názvem "název_adresáře"
rmdir "název_adresáře"	Smaže adresář s názvem "název_adresáře"
ls nebo *ls "název_adresáře"*	Vypíše aktuální adresář nebo vypíše adresář s názvem "název_adresáře"
cd "název_adresáře"	Změní aktuální adresář na adresář s názvem "název_adresáře"
touch "název_souboru"	Vytvoří soubor s názvem "název_souboru"
rm "název_souboru"	Smaže soubor s názvem "název_souboru"
upload "l_s""s_s"	Nahraje lokální soubor s názvem "l_s"na server do souboru s názvem "s_s"
download "s_s""l_s"	Stáhne soubor ze serveru s názvem "s_s"do lokálního souboru s názvem "l_s"
mv "z_s""c_s"	Přesune soubor ze souboru s názvem "z_s"do souboru s názvem "c_s"

Tabulka 3.1: Přehled implementovaných požadavků.

3.6 Zhodnocení implementace

Jak již bylo řečeno systém KIVFS2 vychází ze systému KIVFS a opravuje jeho nedostatky. Hlavním cílem implementace bylo provedení refaktoringu

serverové části. Oproti systému KIVFS je rozdělen systém KIVFS2 pouze do tří vrstev. Vrstvy již nejsou realizovány jako nezávislé moduly, ale každá vrstva běží ve vlastním vlákně. Databázová vrstva byla v systému KIVFS2 integrována do vrstvy Synchronizační. Tím došlo k úspoře komunikace. Nově byla přidána možnost provádět komunikaci mezi vrstvami pomocí unix soketů. Touto změnou bylo docíleno zvýšení výkonu o **+38,2%**, viz tab. 2.1.

Do systému KIVFS2 byly implementovány algoritmy pro mechanizmy, které jsou nezbytné pro správné fungování distribuovaného souborového systému. K zajištění bezpečnosti při přenosu dat, je použito šifrování. Toto šifrování využívá protokol SSL. Dále proti neoprávněnému přístupu do systému je použit proces autentizace a autorizace. Autentizace je založená na protokolu Kerberos. Autorizace probíhá na základě ověření uloženého v databázi. Systém KIVFS2 používá MySQL jako databázový systém. KIVFS2 uchovává většinu informací ve formě metadat které jsou také uloženy v databázi. Komunikaci s ním realizuje synchronizační vrstva. Tato vrstva také zajišťuje synchronizaci požadavků na základě Lamportových logických hodin. KIVFS2 má implementovaný mechanizmus pro dynamický routing. Aby bylo možné používat dynamický routing je v systému použit Dijkstrův algoritmus pro hledání nejrychlejších cest mezi všemi uzly. Tyto nejrychlejší cesty jsou uloženy na každém uzlu a jsou použity pro směrování požadavku. Nedílnou součástí, pro směrování požadavků po nejrychlejších cestách, je implementace mechanizmu pro tunelové spojení mezi uzly. Tunelové spojení je použito jak pro směrování požadavků, tak pro přenos souborů. Fyzické soubory jsou v systému KIVFS2 ukládány po částech stejných velikostí na lokálním souborovém systému daného uzlu. Touto variantou je docíleno efektivnějšího způsobu replikování dat. KIVFS2 používá replikace typu master-master, tedy každý uzel může číst i zapisovat. Aby bylo možné používat tento typ replikací je nezbytné dodržet alespoň sekvenční konzistenci. Z tohoto důvodu jsou v systému zavedeny algoritmy pro synchronizaci a distribuované provádění požadavků. Replikace požadavků probíhá vždy při provádění daného požadavku. Replikace fyzických dat probíhá asynchronně při uploadu souboru. KIVFS2 má implementovaný mechanizmus pro upload a download souborů. Oba přenosy jsou realizovány na bázi pasivního režimu FTP.

V rámci implementace byl vytvořen i klient, který slouží jako testovací nástroj pro ověření správné funkčnosti serveru. Všechny požadavky, které je KIVFS2 schopný zpracovat, jsou uvedeny v tab. 3.1.

4 Testování

Testování bylo realizováno z důvodu ověření, jestli došlo k zvýšení výkonu nově implementovaného systému oproti předchozímu. Výsledky těchto testů byly porovnány s předchozí verzí systému KIVFS. Dále byly provedeny jednotkové testy, kvůli ověření veškeré funkčnosti nově implementovaného systému.

4.1 Unit testy

Pro ověření implementované funkcionality byl použit unit testing. Všechny implementované požadavky jsou uvedeny v tab. 3.1. K unit testingu byla použita knihovna *Check*[4]. Provedené testy a jejich výsledky jsou uvedené v tab. 4.1.

Provedený test	Výsledek	Provedený test	Výsledek
Vytvoření složky	Úspěch	Změna adresáře, který neexistuje	Úspěch
Vytvoření složky, který neexistuje	Úspěch	Nahrání souboru, co ještě neexistuje	Úspěch
Smazání složky	Úspěch	Nahrání souboru, co už existuje	Úspěch
Smazání složky, který neexistuje	Úspěch	Nahrání souboru, skrz tunel	Úspěch
Vytvoření souboru	Úspěch	Stažení souboru	Úspěch
Vytvoření souboru, který neexistuje	Úspěch	Stažení neexistujícího souboru	Úspěch
Smazání souboru	Úspěch	Stažení souboru, skrz tunel	Úspěch
Smazání souboru, který neexistuje	Úspěch	Výpis složky	Úspěch
Použití příkazu {rmdir} na soubor	Úspěch	Použití příkazu {ls} na soubor	Úspěch
Použití příkazu {rm} na složku	Úspěch	Přesunutí souboru	Úspěch
Změna adresáře	Úspěch	Přesunutí souboru, když neexistuje	Úspěch

Tabulka 4.1: Provedené unit testy.

4.2 Testovací prostředí

Pro testování byli vyhrazeny čtyři fyzické servery se shodnou konfigurací, viz. tab. 4.2. Na třech serverech běžel server a na jednom byl puštěný klient systému KIVFS2.

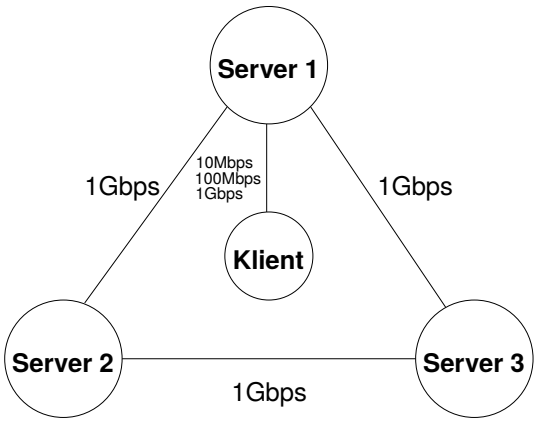
4.3 Upload - přímo

V tomto testu byla ověřována rychlost přenosu dat v systému KIVFS2 a porovnána s jeho starší verzí[25]. Při testování nebylo povoleno dynamické

OS	Debian GNU/Linux 8 (jessie)
Procesor	Intel(R) Pentium(R) D CPU 3.40GHz
Paměť	512 MB
Síť	1Gbps

Tabulka 4.2: Testovací prostředí.

směrování. Topologie, která byla použita pro testování, je na obr. 4.1. Servery jsou propojeny linou o rychlosti 1Gbps. Samotný test měřil rychlost uploadu dat o velikosti 10MB, 100MB a 1GB. Pro každou velikost bylo provedeno deset opakování. Výsledky testu klienta, který byl připojen rychlostí 10Mbps, 100Mbps a 1Gbps jsou zaznamenány v tab. 4.3.



Obrázek 4.1: Topologie KIVFS2 pro přímý upload a download.

		Rychlost linky 10 Mbps			Rychlost linky 100 Mbps			Rychlost linky 1000 Mbps		
Velikost souboru		KIVFS	KIVFS2		KIVFS	KIVFS2		KIVFS	KIVFS2	
10 MB	Čas (s)	00:10,3	00:10,2		00:02,5	00:01,35		00:01,6	00:00,45	
	Rychlost (Mbps)	0,97	**0,98**	+1%	4,08	**7,41**	+81%	6,45	**22,2**	+244%
100 MB	Čas (s)	01:31,2	01:38,2		00:10,3	00:10,5		00:04,1	00:02,2	
	Rychlost (Mbps)	1,10	1,01	-9%	9,71	9,5	-3%	24,27	**45,45**	+87%
1000 MB	Čas (s)	14:53,4	16:08,5		01:31,0	01:39,6		00:26,3	00:20,25	
	Rychlost (Mbps)	1,12	1,03	-9%	10,98	10,04	-9%	38,01	**49,38**	+30%

Tabulka 4.3: Výsledky prvního testovacího scénáře.

Z výsledku je patrné, že nově navržený systém dosahuje lepších výsledků na lince o rychlosti 1Gbps. Na druhou stranu zaznamenává zhoršení na linkách o rychlosti 10Mbps a 1000Mbps.

Protože výsledky nevyšli úplně podle předpokladů, je nezbytné prověřit důkladněji prostředí, na kterém se testuje, jestli je vše v pořádku. Při testování uploadu může docházet k problému při přenosu dat nebo při přístupu na disk. Pro ověření správné funkčnosti linky se využije nástroje iPerf[20].

Linka	Přenos	Šířka pásma
1000Mbps	1.10 GBytes	944 Mbits/sec
100Mbps	97.1 MBytes	81.3 Mbits/sec
10Mbps	10.1 MBytes	8.29 Mbits/sec

Tabulka 4.4: Výsledky měření šířky pásma.

Jedná se o nástroj, který měří maximální dosažitelnou šířku pásma. Výsledky z provedeného testování nástrojem iPerf jsou uvedeny v tab. 4.4.

Dále byl proveden zátěžový test linky:

dd if=/dev/zero bs=1M count=1000000 | pv -b -a | nc 147.228.67.122 10000
3.29GiB [112MiB/s]

Při zátěžovém testu se rychlost držela na 112MiB/s, což +- odpovídá 1Gbps lince. Z výsledků nástroje iPerf a z výsledků zátěžového testu je patrné že testovaná linka je v pořádku.

Pro ověření výkonosti disku byl proveden test pomocí utility hdparm:

hdparm -Tt /dev/sda
/dev/sda:
Timing cached reads: 1900 MB in 2.00 seconds = 949.91 MB/sec
Timing buffered disk reads: 202 MB in 3.02 seconds = 66.87 MB/sec

Z tohoto testu vyplívá, že disk má negativní vliv na provedené testování uploadu souboru. Z tohoto důvodu byl realizován ještě jeden test, kde se použitý disk nahradí ramdiskem. Výsledky tohoto testu jsou uvedeny v tab. 4.5.

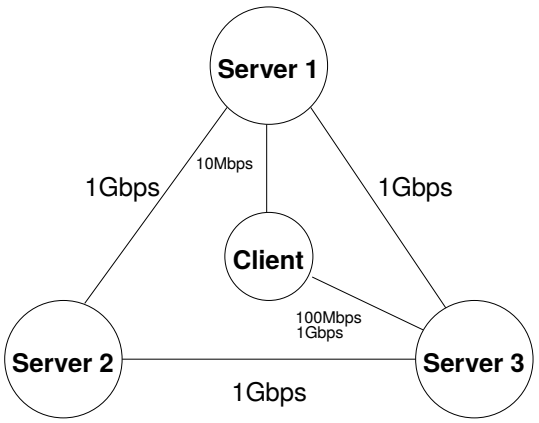
		Rychlost linky 10 Mbps			Rychlost linky 100 Mbps			Rychlost linky 1000 Mbps		
Velikost souboru		KIVFS	KIVFS2		KIVFS	KIVFS2		KIVFS	KIVFS2	
10 MB	Čas (s)	00:10,3	00:10,2		00:02,5	00:01,2		00:01,6	00:00,3	
	Rychlost (Mbps)	0,97	**0,98**	+1%	4,08	**8,54**	+109%	6,45	**33,33**	+416%
100 MB	Čas (s)	01:31,2	01:30,0		00:10,3	00:10,3		00:04,1	00:01,2	
	Rychlost (Mbps)	1,10	**1,11**	+1%	9,71	9,71	0%	24,27	**83,33**	+243%
1000 MB	Čas (s)	14:53,4	11:42,0		01:31,0	01:10,9		00:26,3	00:09,2	
	Rychlost (Mbps)	1,12	**1,42**	+27%	10,98	**14,1**	+28%	38,01	**108,7**	+186%

Tabulka 4.5: Výsledky prvního testovacího scénáře s použitím ramdisku.

Test s použitím ramdisku dopadl podle předpokladů a je z něj patrné zlepšení. Nejlepší výsledky oproti KIVFS dosahuje KIVFS2 na lince o rychlosti 1Gbps.

4.4 Upload - tunel

V druhém scénáři se opět testovala rychlost přenosu dat, ale s povoleným dynamickým směrováním. Použitá topologie je na obr. 4.2. Servery jsou opět propojeny linkou o rychlosti 1Gbps. U klienta přibylo další alternativní připojení o rychlosti 100Mbps a 1Gbps. Opět se testovalo na datech o velikosti 10MB, 100MB a 1GB. Vždy bylo provedeno deset opakování pro každou velikost. Systém KIVFS detekuje přítomnost rychlejšího spojení a skrz něj bude uploadovat data. Výsledky tohoto testu jsou zaznamenány a porovnány s výsledky starší verze[25] v tab. 4.6.



Obrázek 4.2: Topologie KIVFS2 pro upload a download s použitím tunelu.

		Rychlost linky 10 Mbps			Rychlost linky 100 Mbps			Rychlost linky 1000 Mbps		
Velikost souboru		KIVFS	KIVFS2		KIVFS	KIVFS2		KIVFS	KIVFS2	
10 MB	Čas (s)	00:10,5	00:10,3		00:02,5	00:01,2		00:01,6	00:00,5	
	Rychlost (Mbps)	0,95	**0,97**	+2%	4,07	**8,54**	+109%	6,29	**20,2**	+221%
100 MB	Čas (s)	01:31,5	01:38,4		00:10,5	00:10,7		00:04,5	00:02,5	
	Rychlost (Mbps)	1,09	1,01	-8%	9,50	9,3	-3%	22,47	**40,1**	+78%
1000 MB	Čas (s)	14:53,7	16:08,7		01:31,1	01:39,7		00:27,0	00:20,6	
	Rychlost (Mbps)	1,12	1,03	-9%	10,98	10,03	-9%	37,04	**48,54**	+31%

Tabulka 4.6: Výsledky druhého testovacího scénáře.

Testování s použitím tunelů, dopadlo podle očekávání. Došlo k nepatrnému zhoršení oproti testování bez použití tunelů. Toto zhoršení je zapříčiněno dodatečnou režií pro vytváření tunelů. Opět z testování vyplívá, že systém KIVFS2 dosahuje lepších výsledků na lince 1Gbps.

4.5 Download - přímo

Cílem toho testu bylo změřit přenosovou rychlost při downloadu dat v systému KIVFS2. Download souborů probíhá přímo ze serveru, ke kterému je

uživatel připojený. Topologie, která byla použita pro testování, je shodná s topologií pro přímý upload a je zobrazena na obr. 4.1. Servery jsou propojeny linkou o rychlosti 1Gbps. Samotný test měřil rychlost downloadu dat o velikosti 10MB, 100MB a 1GB. Pro každou velikost bylo provedeno deset opakování a každá velikost byla stahována na klientovi, který byl připojen rychlostí 10Mbps, 100Mbps a 1Gbps. Výsledky tohoto testu jsou zaznamenány v tab. 4.7.

		Rychlost linky 10 Mbps		Rychlost linky 100 Mbps		Rychlost linky 1000 Mbps	
Velikost souboru		KIVFS	KIVFS2	KIVFS	KIVFS2	KIVFS	KIVFS2
10 MB	Čas (s)		00:11,0		0:00:02		00:00,5
	Rychlost (Mbps)		0,91		4,43		17,69
100 MB	Čas (s)		01:45,1		0:00:12		00:02,3
	Rychlost (Mbps)		0,95		8,54		43,47
1000 MB	Čas (s)		17:31,3		0:01:46		00:18,4
	Rychlost (Mbps)		0,95		9,41		54,47

Tabulka 4.7: Výsledky třetího testovacího scénáře.

Výsledky pro systém KIVFS nejsou k dispozici, protože se testování na download neprovádělo. Podle očekávání, download dosahuje nejvyšších rychlostí na lince o rychlosti 1Gbps.

4.6 Download - tunel

Stejně jako v testování uploadu, tak i u downloadu bylo provedeno testování s povoleným dynamickým routingem. Cílem toho testu bylo opět změřit přenosovou rychlost při downloadu dat v systému KIVFS2. Download souborů probíhá, pro 10Mbps, přímo ze serveru, ke kterému je uživatel připojený. Pro vyšší rychlosti se v topologii nachází alternativní připojení. Topologie, zvolená pro testování, je opět shodná s topologií pro upload s použitím tunelů a je zobrazena na obr. 4.2. Servery jsou propojeny linkou o rychlosti 1Gbps. Samotný test měřil rychlost downloadu dat o velikosti 10MB, 100MB a 1GB. Pro každou velikost bylo provedeno deset opakování. Výsledky tohoto testu jsou zaznamenány v tab. 4.8.

		Rychlost linky 10 Mbps		Rychlost linky 100 Mbps		Rychlost linky 1000 Mbps	
Velikost souboru		KIVFS	KIVFS2	KIVFS	KIVFS2	KIVFS	KIVFS2
10 MB	Čas (s)		00:11,1		0:00:02,7		00:00,6
	Rychlost (Mbps)		0,90		3,66		16,7
100 MB	Čas (s)		01:46,2		0:00:12,1		00:02,4
	Rychlost (Mbps)		0,94		8,26		41,46
1000 MB	Čas (s)		17:32,0		0:01:46,9		00:18,7
	Rychlost (Mbps)		0,95		9,35		53,4

Tabulka 4.8: Výsledky čtvrtého testovacího scénáře.

Výsledky pro systém KIVFS nejsou k dispozici, protože se testování na download neprovádělo. Z výsledků je patrné, že opět došlo ke zhoršení oproti přímému downloadu pro připojení rychlostí 100Mbps a 1Gbps. Stejně jako u uploadu je to zapříčiněno dodatečnou režií pro vytváření tunelů.

4.7 Upload – 100 souborů

Posledním testem, bylo nahrát na server 100 souborů o velikosti 1MB. Cílem bylo zjistit režii. Použitá topologie je opět shodná s přímým uploadem a je zobrazena na obr. 4.1. Měření probíhalo na rychlostech 10Mbps, 100Mbps a 1Gbps a bylo provedeno deset opakování pro každou rychlost. Servery mezi sebou jsou propojeny linkou o rychlosti 1Gbps. Výsledky tohoto testu jsou zaznamenány v tab. 4.9.

		Rychlost linky 10 Mbps		Rychlost linky 100 Mbps		Rychlost linky1000 Mbps	
Velikost souboru		KIVFS	KIVFS2	KIVFS	KIVFS2	KIVFS	KIVFS2
100 * 1MB	Čas (s)		02:02,8		00:37,4		00:16,4
	Rychlost (Mbps)		0,81		2,67		6,09

Tabulka 4.9: Výsledky pátého testovacího scénáře.

Provedený test potvrdil negativní vliv režie na přenášený soubor, která vzniká při nutnosti otevírat a zavírat každý soubor.

5 Závěr

Cílem této diplomové práce bylo reimplemetovat serverovou část systému KIVFS. Důvodem reimplementace byl ne udržovatelný stav aktuálního systému a technologický posun. Na základě znalostí získaných z předchozích prací by tedy měl být vytvořen nový systém s názvem KIVFS2. Tento refaktoring povede ke zvýšení výkonu a stability nového distribuovaného souborového systému.

Pro potřeby testování nově implementované funkcionality, bylo nezbytné vytvořit testovacího klienta. Tento klient slouží pro otestování implementované funkcionality a pro provedení zátěžových testů. Výsledky zátěžových testů, byli porovnány s výsledky zátěžových testů v systému KIVFS.

Zadání se podařilo splnit v plném rozsahu. Nejprve proběhla analýza starého systému, ze kterého má nový systém vycházet. Dále byla navržena nová architektur, kde došlo ke snížení počtu serverových vrstev z 5 na 3, což vedlo k úspoře komunikace. Jednotlivé vrstvy již nejsou spouštěny jako samostatné moduly, ale jedná se o jediný program, ve kterém každá vrstva běží ve svém vlákně. KIVFS2 nově umožnuje použití unix domain soketů pro komunikaci mezi jednotlivými vrstvami. Použití těchto soketů, místo network soketů, a provedeným refaktoringem došlo ke zvýšení rychlosti komunikace o **38,2%**.

KIVFS2 využívá, pro uchovávání metadat, databázový systém MySQL. Systém MySQL byl zvolen na základě rychlosti práci s daty. Pro autentizaci je použit protokol Keberos, konkrétně jeho implementace s názvem Heimdal. Pro zabezpečení komunikace je použit protokol SSL.

Pro otestování implementované funkčnosti byl použit unit testing, s využitím knihovny *Check*. Bylo provedeno celkem pět zátěžových testů. Jednalo se o tři testovací scénáře pro upload a testovací scénáře pro download. Testováno bylo nahrání souboru přímo na konkrétní server, nahráni souboru s využitím tunelového spojení, stažení souboru přímo z konkrétního serveru, stažení souboru s využitím tunelového spojení a nahrání velkého počtu malých souborů na konkrétní server. Podrobnosti o provedeném testování jsou uvedeny v kapitole 4.

Výsledky ukázali, že nově implementovaný systém KIVFS2 dosahuje lepších výsledků na lince o rychlosti 1Gbps než systém KIVFS. Při uploadu došlo ke zlepšení o **+416%** při nahráváni souboru o velikosti 10MB, o **+243%** při souboru o velikosti 100MB a o **+186%** při přenosu 1GB souboru.

Ve vývoji KIVFS2 lze dále pokračovat v dalších semestrálních, baka-

lářských nebo diplomových prací. Tyto práce by měli být věnované implementaci plnohodnotných klientů pro různé platformy. Zaměřovat by se měli hlavně na mobilní zařízení, které nabízí velký potenciál z hlediska využití systému KIVFS2.

Literatura

[1] Andrew S. Tanenbaum, M. V. S. *Distributed Systems (2nd Ed.): Principles and Paradigms*. Pearson Education. Inc., 2007. ISBN 0-13-239227-5.

[2] *Arla* [online]. [cit. 2017/03/09]. Dostupné z: `http://www.stacken.kth.se/project/arla/`.

[3] Bryla, B. *Oracle Database 12c The Complete Reference*. McGraw-Hill Education, 2013. ISBN 0071801758.

[4] *Check* [online]. [cit. 2017/03/09]. Dostupné z: `https://libcheck.github.io/check/`.

[5] Comeau, A. *MySQL Explained: Your Step By Step Guide*. CreateSpace Independent Publishing Platform, 2015. ISBN 151942437X.

[6] Cristian, F. et al. *Atomic broadcast: From simple message diffusion to Byzantine agreement*. Citeseer, 1986.

[7] *Dijkstras algorithm* [online]. [cit. 2017/03/09]. Dostupné z: `http://www.geeksforgeeks.org/greedy-algorithms-set-6-dijkstras-shortest-path-algorithm/`.

[8] Eliezer Levy, A. S. Distributed File Systems: Concepts and Examples. *ACM Computing Surveys*. December 1990, 22, 4, s. 321–374. ISSN 0360-0300. doi: 10.1145/366622.366644. Dostupné z: `http://www.cs.virginia.edu/~zaher/classes/CS656/levy.pdf`.

[9] Fazekaš, J. *KIVFS: Klient pro GNU/Linux s pomocí jaderného modulu*. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky, 2010.

[10] Freier, A. – Karlton, P. – Kocher, P. The secure sockets layer (SSL) protocol version 3.0. 2011.

[11] Garman, J. *Kerberos: The Definitive Guide*. O'Reilly Media, 2003. ISBN 0596004036.

[12] Glass, E. The NTLM authentication protocol and security support provider. *SourceForge. net*. 2003.

[13] Goldberg, A. V. – Radzik, T. A heuristic improvement of the Bellman-Ford algorithm. *Applied Mathematics Letters*. 1993, 6, 3, s. 3–6.

[14] Hougardy, S. The Floyd–Warshall algorithm on graphs with negative cycles. *Information Processing Letters*. 2010, 110, 8-9, s. 279–281.

[15] Hovorka, K. *KIVFS - Webový klient*. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky, 2010.

[16] Howard, J. H. – others. *An overview of the andrew file system*. Carnegie Mellon University, Information Technology Center, 1988.

[17] Howes, T. – Smith, M. *LDAP: programming directory-enabled applications with lightweight directory access protocol*. Sams Publishing, 1997.

[18] *ini* [online]. [cit. 2017/03/09]. Dostupné z: `https://github.com/rxi/ini`.

[19] *The InterMezzo File System* [online]. [cit. 2017/03/09]. Dostupné z: `https://www.cs.cmu.edu/~coda/docdir/intermezzo99.pdf`.

[20] *iPerf* [online]. [cit. 2017/03/09]. Dostupné z: `https://iperf.fr/`.

[21] Jaroš, P. *KIVFS - Klient pro GNU/Linux v FUSE*. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky, 2010.

[22] Jaroš, P. *KIVFS - Klient pro GNU/Linux s použitím FUSE*. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky, 2012.

[23] Junák, M. *KIVFS: Replikace, synchronizace*. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky, 2010.

[24] M. Wiesmann, F. P. Understanding Replication in Databases and Distributed Systems. April 2000, s. 11. ISSN 1063-6927. doi: 10.1109/ICDCS.2000.840959. Dostupné z: `https://doi.org/10.1109/ICDCS.2000.840959`.

[25] Matějka, L. et al. Dynamic routing in distributed file system. *Journal of Networks*. 2014, 9, 10, s. 2591.

[26] *MS SQL* [online]. [cit. 2017/03/09]. Dostupné z: `https://www.microsoft.com/sql-server`.

[27] Nowicki, B. Nfs: Network file system protocol specification. Technical report, 1989.

[28] *The Network Time Protocol* [online]. [cit. 2017/03/09]. Dostupné z: `https://www.eecis.udel.edu/~mills/ntp/html/index.html#intro`.

[29] Obe, R. *PostgreSQL: Up and Running: A Practical Introduction to the Advanced Open Source Database*. O'Reilly Media, 2014. ISBN 1449373194.

[30] *OpenAFS* [online]. [cit. 2017/03/09]. Dostupné z: `https://www.openafs.org/`.

[31] Pivnička, M. *KIVFS: Zabezpečení, šifrování a ověření identity*. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky, 2009.

[32] Postel, J. User datagram protocol. RFC 768, 1980.

[33] Postel, J. Transmission control protocol. RFC 793, 1981.

[34] Postel, J. – Reynolds, J. File transfer protocol. RFC 765, 1985.

[35] Satyanarayanan, M. Coda: a highly available file system for a distributed workstation environment. August 2002, s. 4. doi: 10.1109/WWOS.1989.109279. Dostupné z: `https://doi.org/10.1109/WWOS.1989.109279`.

[36] Skiena, S. Dijkstra's algorithm. *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica, Reading, MA: Addison-Wesley*. 1990, s. 225–227.

[37] Skupa, J. *KIVFS - Synchronizace a trasování požadavků*. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky, 2012.

[38] Soltis, S. R. – Ruwart, T. M. – OKeefe, M. T. The global file system. 1996.

[39] Stallman, R. M. – McGrath, R. – Smith, P. D. *GNU Make: A program for directing recompilation, for version 3.81*. Free Software Foundation, 2004.

[40] Strejc, R. *KIVFS: Souborový a databázový server*. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky, 2009.

[41] Strejc, R. *KIVFS - Datové uložiště*. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky, 2012.

[42] Vlček, K. *KIVFS: Zabezpečení serveru*. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky, 2014.

[43] Weil, S. A. et al. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th symposium on Operating systems design and implementation*, s. 307–320. USENIX Association, 2006.

[44] Weil, S. A. et al. CRUSH: Controlled, scalable, decentralized placement of replicated data. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, s. 122. ACM, 2006.

Seznam použitých zkratek

Zkratka	**Celý název**
LDAP	Lightweight Directory Access Protocol
NTLM	NT Lan Manager
KDC	Key Distribution Center
AS	Authentication Server
TGS	Ticket Granting Service
ACL	Access Control List
AFS	Andrew File Systém
FIFO	First in first out
NTFS	New Technology File Systém
FAT	File Allocation Table
DBMS	Database Management Systém
API	Application Programming Interface
BSD	Barkeley Software Distribution
MIT	Massachusetts Institute of Technology
SQL	Structured Query Language
NIST	National Institute of Standards and Technology
NTP	Network Time Protocol
LAN	Local Area Network
WAN	Wide Area Network
GFS	Global File Systém
TCP	Transmission Control Protocol
FUSE	Filesystem in userspace
UDP	User Datagram Protocol
SSL	Secure Socket Layer

Tabulka 5.1: Seznam použitých zkratek.

Seznam obrázků

Seznam tabulek

Přílohy

```
[libdefaults]
    default_realm = DFS.ZCU.CZ

# The following krb5.conf variables are only for MIT Kerberos.
    krb4_config = /etc/krb.conf
    krb4_realms = /etc/krb.realms
    kdc_timesync = 1
    ccache_type = 4
    forwardable = true
    proxiable = true

# The following libdefaults parameters are only for Heimdal Kerberos.
    v4_instance_resolve = false
    v4_name_convert = {
        host = {
            rcmd = host
            ftp = ftp
        }
        plain = {
            something = something-else
        }
    }
    fcc-mit-ticketflags = true

[realms]
    TEST.LOCAL = {
        kdc = ul403ds10-kiv.fav.zcu.cz
        admin_server = ul403ds10-kiv.fav.zcu.cz

    }

[domain_realm]
    .fav.zcu.cz = DFS.ZCU.CZ
    fav.zcu.cz = DFS.ZCU.CZ
[login]
    krb4_convert = true
    krb4_get_tickets = false
```

Obrázek 5.1: Ukázka konfiguračního souboru Kerberos serveru.

```
[wrapper]
net_ip=147.228.67.121          - ip adresa, na které server přijímá požadavky od klienta
net_port=10000                 - port, na kterém server přijímá požadavky od klienta
net_encoding=1                 - šifrování komunikace s klientem. 0 - bez šifrování, 1 - šifrování
severity_level=6               - úroveň logování
ip=127.0.0.1                   - ip adresa pro komunikaci s dalšími vrstvami
port=10001                     - pro komunikaci s dalšími vrstvami
encoding=0                     - šifrování komunikace s ostatními vrstvami

[auth]
ip=127.0.0.1                   - ip adresa pro komunikaci s dalšími vrstvami
port=10002                     - pro komunikaci s dalšími vrstvami
encoding=1                     - šifrování komunikace s ostatními vrstvami
hostname=kivfs                 - kerberos hostname
sname=dfs                      - kerberos sname
kt_pathname=/etc/krb5.keytab   - kerberos cesta k souboru keytab

[sync]
ip=127.0.0.1                   - ip adresa pro komunikaci s dalšími vrstvami
pathname=sync                  - jméno unix soketu, pro komunikaci s dalšími vrstvami
encoding=1                     - šifrování komunikace s ostatními vrstvami
node_port=10005                - port na kterém, synchronizační vrstva přijímá systémové požadavky
node_encoding=1                - šifrování komunikace pro systémové požadavky

[fs]
ip=127.0.0.1                   - ip adresa pro komunikaci s dalšími vrstvami
port=10004                     - pro komunikaci s dalšími vrstvami
encoding=1                     - šifrování komunikace s ostatními vrstvami


[node]
ip=147.228.67.121              - ip adresa, na které server přijímá požadavky od ostatních uzlů
port=10010                     - pro komunikaci s dalšími uzly
encoding=0                     - šifrování komunikace s ostatními uzly
id=1                           - id uzlu

[1]
ip=147.228.67.122              - ip adresa, na které server přijímá požadavky od ostatních uzlů
port=10010                     - pro komunikaci s dalšími uzly
encoding=0                     - šifrování komunikace s ostatními uzly
id=2                           - id uzlu

[2]
ip=147.228.67.123              - ip adresa, na které server přijímá požadavky od ostatních uzlů
port=10010                     - pro komunikaci s dalšími uzly
encoding=0                     - šifrování komunikace s ostatními uzly
id=3                           - id uzlu

[db]
username=kivfs                 - jméno uživatele pro přístup do databáze
password=kivfs                 - heslo uživatele pro přístup do databáze
database=KIVFS                 - název databáze

[tunnel]
tunnel=0                       - povolení tunelového spojení. 0 - nepoužívat, 1 - používat
```

Obrázek 5.2: Ukázka konfiguračního souboru serveru.