Západočeská univerzita v Plzni
Fakulta aplikovaných věd

DISERTAČNÍ PRÁCE

2017

Ing. Luboš Matějka

Západočeská Univerzita v Plzni

Fakulta aplikovaných věd

Dynamické směrování v distribuovaných souborových systémech

Ing. Luboš Matějka

disertační práce
k získání akademického titulu "Doktor"
v oboru Informatika a výpočetní technika

Školitel: prof. Ing. Jiří Šafařík, CSc.
Katedra: Informatiky a výpočetní techniky

Plzeň 2017

University of West Bohemia in Pilsen
Faculty of Applied Sciences

Dynamic routing in distributed file system

Ing. Luboš Matějka

Ph.D. thesis
to obtain the academic title "Doctor"
in specialization Computer Science and Engineering

Supervisor: prof. Ing. Jiří Šafařík, CSc.
Department: Computer Science and Engineering

Pilsen 2017

Prohlašuji, že jsem tuto disertační práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Západočeská Univerzita v Plzni má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ..... Podpis autora

Obsah

1. Úvod

1.1 Motivace

Výpočetní technika v podobě počítačů či notebooků se již před několika roky stala nedílnou součástí našich životů. V posledních letech jde vývoj v oblasti informačních technologií velmi rychle kupředu a počítač či notebook je doplněn o mobilní zařízení jako je chytrý telefon nebo tablet. To jsou ale jen viditelné vazby informačních technologií na běžný život. V současné době je počítač v nějaké formě obsažen ve většině elektronických zařízení jako například televize, přehrávače hudby, ale i ledničky či pračky. Všechny tyto formy informačních technologií mají dvě věci společné. Jednak požadují a vytváří data jako jsou multimediální záznamy či logy provozu, druhak tato data prostřednictvím počítačových sítí sdílejí. Díky zkvalitňování jednotlivých technologií objemy sdílených dat velice rychle narůstají, což je problematické pro mobilní zařízení či obecně pro zařízení komunikující prostřednictvím mobilních sítí, které zatím nedosahují dostatečných parametrů na online přístup ke sdíleným datům. Stejně tak úložná kapacita mobilních zařízení je omezená. Vhodným řešením problému požadavku na online dostupnost velkých dat prostřednictvím počítačových sítí s různými a proměnlivými parametry je distribuovaný souborový systém.

Aktuálně dostupné distribuované souborové systémy jsou navrhovány pro provoz na metalických či dokonce optických sítích, kde přenosová rychlost a propustnost datových linek není problém. U mobilních zařízeních využívajících mobilní připojení jako WiFi, EDGE či LTE je ale situace jiná. Přenosová rychlost těchto technologií je o jeden až několik řádů nižší. Zároveň úložný prostor mobilních zařízení je v porovnání s požadavky na současná data nedostačující. Dalším problémem mobilních zažízení je nižší výkon než u notebooků či počítačů. Je zřejmé, že ideálním řešením bude většinu náročnějších úkonů přesunout na servery a minimalizovat tak práci klienta.

Zároveň je v mobilních sítích běžné, že přístup do lokální sítě, tedy sítě aktuálně používaného poskytovatele pro EDGE, LTE či WiFi je výrazně rychlejší a často i levnější než přístup do dalších sítí mimo aktuální ISP. Pokud by distribuovaný souborový systém měl repliku v síti aktuálního poskytovatele připojení, mohlo by se k datům přistupovat nejlepší možnou rychlostí a za nejnižší možnou cenu. Tento požadavek řeší distribuovaný souborový systém pomocí replikací, což je funkce, kterou většina existujících řešení, jako například OpenAFS[100][101], implementovanou mají, ale pouze v read-only režimu.

Zároveň žádné ze současných řešení zcela nesplňuje požadavky nasazení na mobilních zařízeních a zároveň v geograficky rozsáhlých systémech, což se stále častěji ukazuje jako nutnost, neboť organizace provozující velké datové sklady stále častěji rozmisťují své servery v datacentrech na různých kontinentech, aby tak zvýšili rychlost odezvy provozovaných systémů. Konkrétní nedostatky budou v rámci této práce specifikovány na základě vlastností existujících řešení a teoretických základů fungování distribuovaných systémů.

1.2 Struktura práce

V kapitolách 2 až 6 práce budou představeny jednotlivé současné technologie, kterými se bude tato práce zabývat a budou specifikovány jejich vlastnosti. Nejprve bude v kapitole 2 popsáno současné prostředí mobilních zařízení. Budou představeny jednotlivé typy běžně dostupných mobilních zařízení a budou popsány jejich parametry a omezení, která jsou pro provoz distribuovaného souborového systému podstatná.

V kapitole 3 bude popsáno prostředí a specifika geograficky rozsáhlých systémů, jakožto prostředí, kde jsou typicky distribuované systémy nasazovány a budou popsána omezení, která v geograficky rozsáhlém prostředí vznikají a způsobují komplikace pro chod distribuovaných souborových systémů.

V kapitole 4 budou popsány možnosti směrování požadavků v počítačových sítích, které představují základní možnost optimalizace datových toků v počítačových sítích. Na základě popisu existujících řešení budou popsána omezení, která neumožňují efektivní nasazení dynamických směrovacích protokolů v geograficky rozsáhlých distribuovaných systémech.

Kapitola 5 obsahuje stěžejní obsah teoretické části této práce a sice popis vlastností, fungování a nedostatků existujících distribuovaných souborových systémů. Detailně budou popsány stavební kameny nutné pro návrh nového distribuovaného systému, jako jsou *synchronizace času pomocí logických hodin, replikace dat, konzistence, transparentní přístup klientů k datům.*

V kapitole 6 jsou na základě informací uvedených v předchozích kapitolách specifikovány nedostatky existujících distribuovaných souborových systémů a možnosti nasazení distribuovaných souborových systémů pro mobilní zařízení a zároveň geograficky rozsáhlé sítě. Na základě těchto poznatků jsou specifikovány cíle této práce, které umožní nasazení nového distribuovaného systému pro použití na mobilních zařízeních a zároveň umožní provoz v rozsáhlých sítích.

V kapitole 7 je představen návrh nového distribuovaného souborového systému KIVFS. V této části textu je představena architektura systému KIVFS, jsou posány jednotlivé programové části, které celý systém tvoří a jsou popsány mechanizmy nutné pro fungování distribuovaného souborového systému. V závěrečných podkapitolách kapitoly 7 jsou navrženy a provedeny výkonnostní testy porovnávající nové řešení se současným existujícím systémem OpenAFS. V podkapitolách 7.4 a 7.7 jsou představeny prostředky, které umožňují další optimalizaci fungování systému, které budou detailně popsány v samostatných kapitolách 9.3 a 9.

V kapitole 9.3 je popsán postup zefektivnění replikace dat a navržen systém Multi-Master Online RW replikace, která nejen minimalizuje časovou náročnost replikace jak z pohledu klienta tak z pohledu systému samotného. Navíc díky existenci synchronních dat na více uzlech staví základ pro možnou optimalizaci datových cest.

Kapitola 9 obsahuje návrh a popis implementace optimalizace směrování datových požadavků v distribuovaném souborovém systému bez použití klasických směrovacích protokolů, ale s využitím algoritmů v těchto protokolech využívaných. Závěrem jsou uvedeny výkonnostní testy, které prokazují správnost navrženého systému.

V poslední kapitole 10 je zhodnoceno naplnění jednotlivých cílů na základě výsledků výkonnostních testů pro jednotlivá navržená zlepšení. V krátkosti jsou

představeny další možné cesty optimalizace, které se během implementace objevily.

2. Mobilní zařízení

Mobilní zařízení je pojem, který dnes zahrnuje velké množství různých typů zařízení, které se stále více svojí architekturou přibližují ke klasickému osobnímu počítači, ač jejich primární účel použití je jiný. Mobilní zařízení můžeme historicky rozdělit na tři skupiny. Mobilní počítače, mobilní telefony a tablety.

2.1 Typy mobilních zařízení

Mobilní počítač, nebo také notebook je zařízení, které přímo z osobního počítače vychází a snaží se uživateli umožnit plnohodnotnou práci s PC mimo klasickou kancelář. Oproti osobnímu počítači jsou ale na notebooky kladeny jiné požadavky. Jelikož se jedná o mobilní zařízení je jedním z primárních parametrů nízká hmotnost a malé rozměry. Jelikož mobilní zařízení není trvale připojeno ke zdroji elektrické energie, ale je napájeno baterií, je velmi podstatným parametrem nízká energetická náročnost. Oba tyto požadavky na notebooky sebou jako logický důsledek přinášejí snížení dalších parametrů zařízení, což je nižší výkon a menší úložný prostor.

U mobilního telefonu je situace historicky jiná. Mobilní telefon vychází z klasického telefonu, kde primárním cílem bylo umožnit mobilní volání. První mobilní telefony byla jednoúčelová zařízení, která sloužila jen k volání či posílání krátkých textových zpráv. Postupně se mobilní telefon začal svojí architekturou přibližovat počítači. Zůstala možnost volání, ale už i ta je často realizovaná pomocí mobilního operačního systému, který umožňuje i další funkce jako je prohlížení internetu či práci s elektronickou poštou. Taková zařízení se nyní nazývají chytré telefony. Operační systém chytrých telefonů byl nejprve jednoúčelový, například Symbian[1]. Postupně se na chytré telefony začaly portovat operační systémy postavené na základě klasických desktopových operačních systémů jako je Android[2], iOS[3] či Microsoft Windows[4]. S příchodem těchto operačních systémů na mobilní zařízení vznikl požadavek na fungovaní mobilní kanceláře i z chytrých telefonů a tím i požadavek na přístup ke všem datům prostřednictvím mobilní sítě stejně jako u notebooků.

Poslední skupinou mobilních zařízení jsou tablety. Tablet jako zařízení se z pohledu užití a výkonu nachází mezi notebookem a chytrým telefonem. Tablet disponuje vyšším výkonem a větším displayem než chytrý telefon, ale má stejné možnosti připojení k počítačové síti a podobně velký diskový prostor jako mobilní telefon. Téměř ve všech případech u masivně rozšířených tabletů najdeme stejný operační systém jako na chytrých telefonech a můžeme tedy tato dvě zařízení brát z pohledu možnosti připojení na distribuované úložiště jako zařízení se stejnými omezeními.

Jak bylo uvedeno v předchozích odstavcích, hlavními nedostatky mobilních zařízení jsou nedostatečný výkon, malá dostupná úložná kapacita a omezené prostředky mobilního připojení. Nedostatek úložného prostoru můžeme eliminovat použitím distribuovaného souborové systému, kde kromě zobrazení a modifikace dat přeneseme většinu zátěže, která by mohla výkonově zatěžovat mobilní zařízení, na server. Zůstává ale problém mobilní konektivity.

Notebooky stejně jako osobní počítače či servery disponují možností připojení

ke klasické metalické síti, která ale není dostupná všude. U chytrých telefonů a tabletů taková možnost není. Mobilní zařízení běžně využívají dvou typů přístupu k mobilní síti. Jedním jsou WiFi sítě, druhým sítě vycházející z GSM technologií.

Pro WiFi zařízení existuje velké množství standardů, které se liší ve spolehlivosti, přenosové kapacitě či vzdálenosti, na kterou jsou schopna data přenášet. Primárním účelem WiFi sítí bylo umožnit bezdrátový přístup k počítačové síti v rámci omezeného prostoru například v kanceláři, v rodinném domě či restauraci. Tyto sítě jsou konstruovány jako multipointy, kde k jednomu přístupovému bodu je připojeno více klientů.

Paralelně se vyvíjely varianty umožňující bezdrátové přenosy dat na větší vzdálenosti, kde se spíše jedná o point-to-point spoje, které neslouží k připojení jednotlivých mobilních klientů, ale které slouží k náhradě metalických či optických spojů. Tyto spoje disponují větší přenosovou kapacitou a jsou dostupné na větší vzdálenosti až v řádech kilometrů, ale u mobilních zařízení se díky svým nárokům na energii a přesnost zaměření nepoužívají. Mezi taková technologie patří 802.11a[6].

Naopak přenosové technologie založené na standardech 802.11b,g,n,ac jsou přímo pro mobilní zařízení navržena. Použitelnost konkrétní technologie závisí na dostupnosti na daném zařízení a dostupnosti v dané lokalitě. Přenosové kapacity jednotlivých zařízení jsou uvedena v následující tabulce 2.1.

Název	**Rychlost**
CSD	14,4 kbps
HSCSD	57,6 kbps
ECSD	236 kbps
GPRS	171 kbps
EGPRS	296 kbps
EDGE	384 kbps
UMTS	2 Mbps
HSDPA	10 Mbps
LTE	300 Mbps
802.11a[6]	54 Mbps
802.11b[7]	11 Mbps
802.11g[8]	54 Mbps
802.11n[9]	150 Mbps
802.11ac[10]	800 Mbps

Tabulka 2.1: Maximální rychlosti mobilních přenosových technologií

Jednotlivé vyjmenované technologie jsou v rámci dané skupiny vzájemně zpětně kompatibilní. Tedy například EDGE a GRPS či 802.11 b,g,n,ac. Stejně tak většina dnešních vysílačů je schopna kromě své nejlepší technologie vysílat i ve starších formátech. Jednotlivé varianty jsou také různě závislé na síle signálu. Pro vyšší přenosové rychlosti je třeba vyšší kvality signálu, aby se mohlo použít například více úrovňových stavů a na základě Nyquistova kritéria 2.1 docílit vyšší přenosové rychlosti.

$$c = 2w * \log_2(V)[b/s] \tag{2.1}$$

V rámci vzorce 2.1 je c je kapacita přenosového kanálu, w je šířka pásma a V je počet stavů identifikovatelných pro daný přenosový kanál.

Tento postup se používá například u přenosové technologie EDGE, která se v případě zhoršeného signálu degraduje na výrazně pomalejší GPRS, které však není tak citlivé na kvalitu signálu. Obdobná situace, ale s využitím jiných modulací a více kanálů je u WiFi technologií 802.11. Z výše uvedeného je patrné, že při použití technologií pro mobilní připojení se kvalita a rychlost připojení bude v čase měnit na základě okolností, které nejsme schopni ovlivnit.

2.2 Kapacita mobilních zařízení

Podobně jako je omezena přenosová rychlost mobilních zařízení je omezen i úložný prostor mobilních zařízení. V závislosti na typu mobilního zařízení se dostupná úložná kapacita pohybuje od jednotek MB u mobilních telefonů až k jednotkám TB u notebooků. Základní velikost úložného prostoru, která je dána výrobcem, je u většiny zařízení možné rozšířit o doplňkovou pamět typu SD karty pro mobilní telefony a tablety či USB a flash disky pro notebooky. I s touto přidanou kapacitou se však pohybujeme maximálně v řádech jednotek TB a to jen u několika málo zařízení. Naproti tomu velikost datových skladů se díky obrovskému nárůstu nároků na přesnost datových souborů jako jsou fotografie a s ohledem na stále se snižující cenu pevných disků pohybuje až ve stovkách TB. Taková kapacita není aktuálně pro mobilní zařízení dostupná a to hned z několika důvodů. Prvním důvodem je rozměr takového úložiště, které by znemožňovalo či výrazně snižovalo mobilitu zařízení. Druhým důvodem jsou vysoké energetické nároky takového datového skladu a s tím související potřeba chlazení, které není na mobilních zařízeních, disponujících z důvodu nízké hlučnosti často jen pasivním chlazením, možná.

2.3 Operační systém mobilních zařízení

Mobilní zařízení můžeme podle typu operačního systému dělit na dvě skupiny. První skupinou jsou notebooky, které od počátku svého vzniku používají obdobné operační systémy jako stolní počítače či servery, tedy například Microsoft Windows či GNU/Linux. Druhou mladší skupinou jsou mobilní telefony potažmo tablety, které z mobilních telefonů vznikly. V prvopočátku mobilní telefon neměl operační systém v pravém slova smyslu. Disponoval pouze specifickým softwarem jako byl například Psion z roku 1980, který disponoval pouze omezenou sadou funkcí, kterou nebylo možné doplňovat či modifikovat. To se částečně změnilo v roce 1991 s příchodem operačního systému EPOC. EPOC byl operační systém, do kterého bylo možné doprogramovávat další aplikace a funkce. Ale tento operační systém byl stále uzavřený. První operační systém pro mobilní zařízení, do kterého bylo možné doprogramovávat libovolné aplikace byl představen v roce 2001 a jednalo se o Symbian V60. Díky možnosti doprogramovávat libovolné aplikace se částečně otevřela teoretická možnost připojit k mobilnímu zařízení i například distribuované úložiště. Reálná možnost přišla až s příchodem operačních systémů iOS, Android a Windows Mobile. Tyto operační systémy mají společný základ s operačními systémy z PC či serverů, v případě iOS a Android v Unixu, v pří-

padě Windows Mobile v Microsoft Windows. Platforma Windows Mobile byla postupně opuštěna a nyní je nahrazována operačním systémem Microsoft Windows 10, což opět usnadňuje portování složitějších aplikací na mobilní zařízení. Podle aktuálního rozložení trhu, který je znázorněn v tabulce 2.2, je patrné, že trhu dominují dvě platformy a to iOS a Android a proto parametry a vlastnosti distribuovaného souborového systému budeme konstruovat s ohledem na právě tyto dvě platformy.

Operační systém	**Zastoupení na trhu v roce 2017**
Android	85.0%
iOS	14.7%
Windows Mobile / Windows 10	00.1%
Ostatní	00.2%

Tabulka 2.2: Zastoupení jednotlivých operačních systémů na chytrých telefonech[5]

2.4 Řešení v podobě distribuovaného souborového systému

V současné době je zřejmé, že používání mobilní techniky a tedy i nutnost přistupovat k velkým objemům dat bude nadále narůstat. Navíc je stále nutnější data sdílet ať už v rámci uzavřených organizací, jako jsou firmy nebo univerzity, tak i externě dalším uživatelům. Přestože se vlastnosti mobilních zařízení budou postupně zlepšovat, je zřejmé, že ve stejném či rychlejším tempu bude narůstat velikost dat, která budou pro plnohodnotné využití mobilních zařízení nutná. Jako možné řešení tohoto nedostatku, který brzdí rozvoj plnohodnotného využití mobilních zařízení, se nabízí použití distribuovaného souborového systému. Při použití distribuovaného soubového systému data nemusí být uložena na samotném mobilním zařízení, ale uživatel s nimi i přesto může pracovat jako s lokálními daty (jak bude uvedeno v kapitole 5.2). K datům je možné zabezpečeně přistupovat prostřednictvím libovolné počítačové sítě (jak bude uvedeno v kapitole 5.7). Data mohou být uložena na více serverech, což zvyšuje jejich dostupnost jak z hlediska přístupové rychlosti tak z hlediska odolnosti proti selhání (jak bude uvedeno v kapitole 5.3).

3. Geograficky rozsáhlé systémy

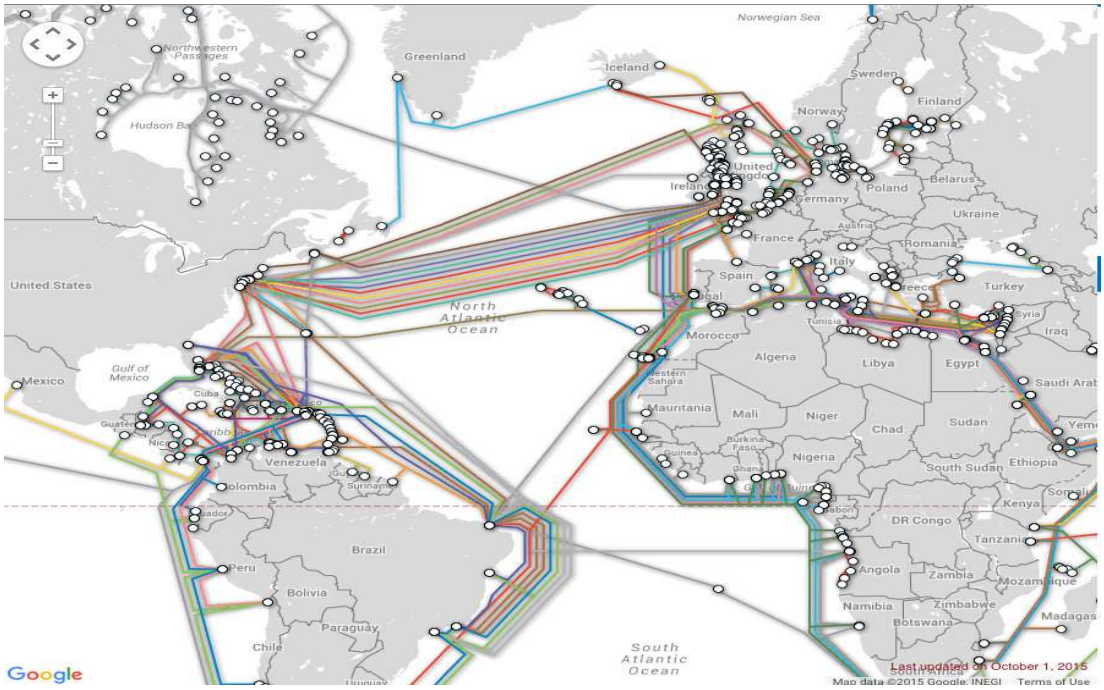
V průběhu studia možností mobilních zařízení, jehož výsledky jsou popsány v kapitole 2 a studia možností a vlastností distribuovaných souborových systémů, jehož výsledky jsou popsány v kapitole 5, se ukázalo, že některá omezení vznikající při práci s počítačovou sítí na mobilních zařízeních, se vyskytují i při práci s počítačovou sítí v rámci distribuovaných souborových systémů. Konkrétně se jedná o zpoždění RTT [24], které se vyskytuje v geograficky rozsáhlých systémech, kde data musejí prostřednictvím počítačové sítě překonávat velké vzdálenosti. Nabízí se tedy otázka, zda tento společný problém nemůže mít společné řešení, které by problém eliminovalo jak pro mobilní zařízení tak i pro distribuovaný souborový systém. Problematika geograficky rozsáhlých systémů bude detailněji popsána v následujících podkapitolách.

3.1 Definice geograficky rozsáhlých systémů

Jednou z požadovaných vlastností distribuovaných systémů, clusterů či cloudů je vysoká dostupnost. Tato vlastnost je typicky řešena pomocí redundance dat a výpočetních uzlů. V prvních implementacích stačilo násobné zapojení uzlů, které v případě výpadku převzaly roli nefunkčního uzlu. Postupně se tento model zlepšoval. K prosté redundanci se přidala redundance napájení, následně redundance konektivity. Oba tyto prvky mají v rámci jedné serverovny či lokality společné omezení. Tím je dočasná nedostupnost lokality při výpadku napájení či konektivity. Budou-li například všechny uzly distribuovaného systému v jedné lokalitě a na této lokalitě dojde k výpadku napájení, bude z globálního pohledu nefunkční celý systém. Stejně tak výpadek konektivity pro danou lokalitu (kterou může být serverovna, ale i celé město či kraj) může způsobit nedostupnost celého systému. Dalším logickým krokem v redundanci je geografické rozmístění uzlů distribuovaného systému. Jednotlivé uzly distribuovaného systému budou rozmístěny do vzájemně nezávislých lokalit s zcela nezávislým řešením konektivity a napájení. Navíc mohou být jednotlivé uzly umístěny na různých kontinentech a to tak, aby v dané lokalitě byly co nejblíže k uživatelům. Díky současným kapacitám přenosových linek mohou být jednotlivé uzly v jedné lokalitě propojeny až 10 či 100Gbps optickými či metalickými linkami. Stejně tak jednotlivé lokality mohou být propojené násobnou konektivitou dosahující rychlostí 100 až 400 Gbps jak je ukázáno na obrázku 3.1 pro celosvětové propojení.

Násobné vysokorychlostní propojení v řádech desítek Gbps už není jen výsadou spojení kontinentů, ale obdobné sítě se budují i na národních úrovních, jak můžeme jako příklad vidět na obrázku 3.2 pro příklad sítě Cesnet pro Českou republiku.

Tato rychlá propojení umožňují snadné budování geograficky oddělených systémů a tím přispívají k větší odolnosti distribuovaných systémů vůči výpadkům. Tím, že distribuovaný souborový systém je pro uživatele z pohledu uložení dat transparentní, jak bude popsáno v kapitole 5.7, je navíc možné nejvíce používaná data danou skupinou uživatelů přemístit do co nejbližšího uzlu a tím docílit co nejrychlejšího a zároveň z hlediska mezinárodních přenosů co nejlevnějšího řešení.



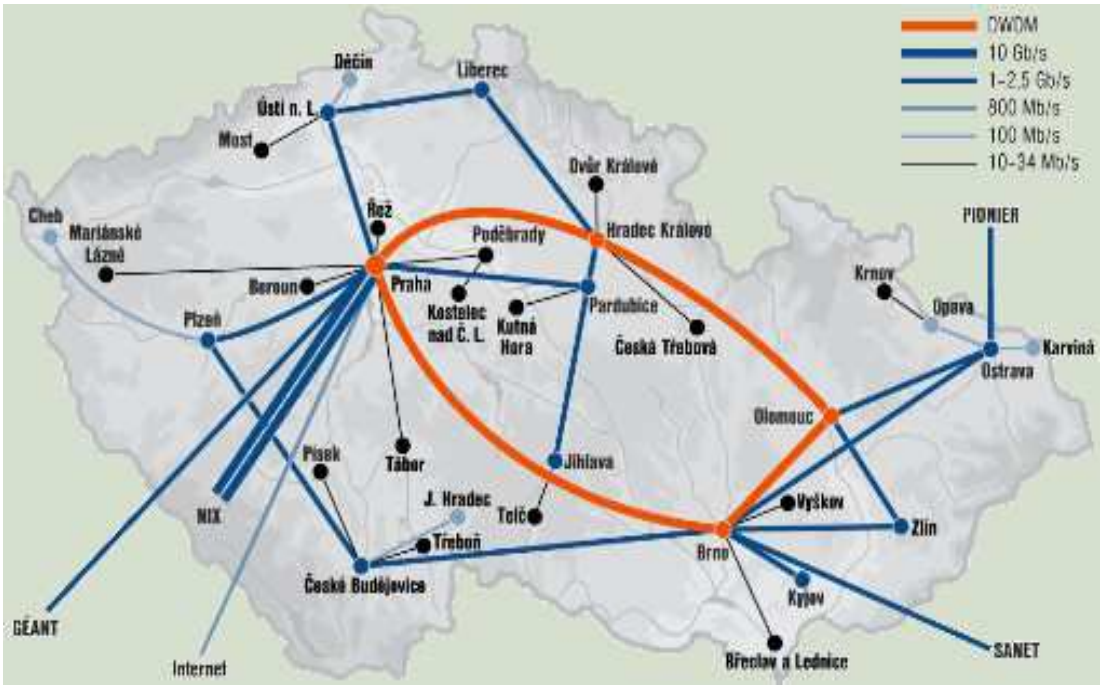
Obrázek 3.1: Ukázka internetového propojení kontinentů, rok 2016, zdroj http://www.submarinecablemap.com

3.2 Omezení geograficky rozsáhlých systémů

Nedostatky mobilních zařízení popsaných v kapitole 2 geograficky rozsáhlé systémy na národní či dokonce mezinárodní úrovni netrpí. Trpí ale jiným, v projevech velmi podobným nedostatkem. Zatím co u mobilních zařízení je limitující přenosová rychlost a potažmo šířka dostupného pásma, u geograficky rozsáhlých systémů je zásadním problémem vzdálenost. Digitální data se v počítačové síti přenášejí analogovým signálem prostřednictvím přenosových kanálů jako je metalický vodič, optické vlákno či někde bezdrátové spojení. Tento analogový přenos má pro různý druh vodiče různou charakteristiku, různý útlum a různou maximální vzdálenost, kterou může dosáhnout, stejně tak jako různou rychlost, jakou je schopen se daným nosičem signálu šířit. Tím, že maximální délka souvislého přenosového média je omezena, jak ukazuje následující tabulka 3.1, je nutné vodič přerušit a doplnit o aktivní prvek jako je router či switch nebo v případě dálkových spojů o opakovač signálu. Ve všech třech případech se však k maximální rychlosti přenosu signálu, která je specifická pro dané přenosové médium a signál, musí připočítat čas nutný k přijetí a znovu vyslání signálu, který se označuje jako RTT.

RTT, narozdíl od TTL, které se typicky řeší u mobilních zařízení, přímo neovlivňuje přenosovou rychlost ve všech případech. Pokud se budou na síti s vyšším RTT a malou chybovostí realizovat souvislé datové přenosy větších datových bloků, bude zpoždění zanedbatelné, neboť na běžně používaných TCP/IP spojení je realizováno odesílání dat s klouzajícím okénkem. Pomalejší bude jen prvotní navázání spojení, ale samotný přenos už tímto zpožděním výrazně omezen nebude.

Zcela jiná situace nastává u přenosů, kde se realizuje přenos velkého množství velmi malých datových bloků, například v jednotkách bytů či kilobytů nebo v případě, že linka vykazuje vysokou chybovost a přenos jednotlivých bloků se musí opakovat. Typickým příkladem přenosu s velkým množstvím přenosu malých da-



Obrázek 3.2: Ukázka zapojení akademické sítě Cesnet, rok 2016, zdroj http://www.cesnet.cz

Typ vodiče	**Délka souvislého vodiče[m]**
tenký koaxiální kabel	200
tlustý koaxiální kabel	500
CAT5e / 1Gbps	100
CAT5e / 10Gbps	55
CAT6 / 10Gbps	55
CAT6a / 10Gbps	100
optický kabel 100Mbps - 100Gbps	70000

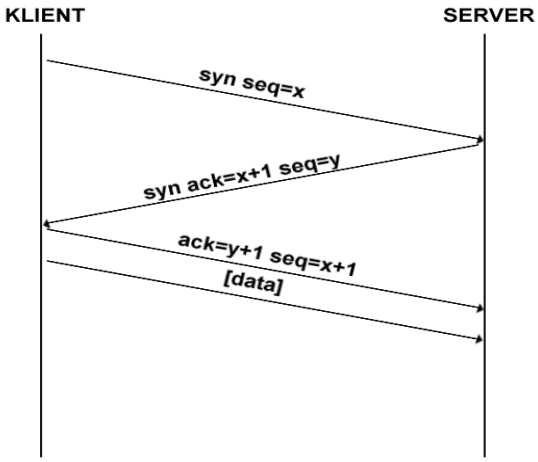
Tabulka 3.1: Maximální souvislé délky jednotlivých typů vedení.

tových bloků je provoz distribuovaného souborového systému. Jak bude uvedeno v kapitole 5.5.2, v rámci distribuovaného souborového systému se o pořadí jednotlivých operací hlasuje a cílem hlasování je docílit pořadí vykonávání operací na jednotlivých uzlech podle jednoho ze zvolených konzistenčních modelů popsaných v kapitole 5.4.2. Při každém požadavku může být v distribuovaném souborovém systému přeneseno takové množství zpráv, které odpovídá dvojnásobku počtu uzlů, neboť uzel inicializující operaci zašle všem ostatním uzlům zprávu s návrhem pořadí operace a všechny uzly zpět zasílají své rozhodnutí pro danou operaci. Situace se může ještě zkomplikovat v případě, že návrh není většinou schválen a pak je třeba celou operaci opakovat. Pro takovýto provoz je každé zpoždění na přenosovém kanálu velice problematické, neboť jednotlivé uzly zprostředkovávají I/O operace, které jsou pro klienty blokující. Tedy dokud hlasování o dané operaci není úspěšně dokončeno, není možné žádnou další operaci provést. Hodnoty RTT pro běžně používané typy datových nosičů a maximální délky těchto vodičů jsou zobrazeny v tabulce 3.2.

Typ vodiče	Délka vodiče[m]	RTT[ns/m]
koaxiální kabel	200m	6.3784
CAT5e / 1Gbps	100	5.76
CAT5e / 10Gbps	55	5.76
CAT6 / 10Gbps	55	5.76
CAT6a / 10Gbps	100	5.76
optický kabel 100Mbps - 100Gbps	70000	4.90

Tabulka 3.2: Hodnoty RTT pro běžně používané typy přenosových médií

Hodnoty RTT pohybující se v jednotkách ns/m nevypadají na první pohled pro rychlost a plynulost datových přenosů problematicky. Je ale třeba si uvědomit, že u geograficky rozsáhlých systémů mohou jednotlivé uzly dělit vzdálenosti jednotek nebo dokonce desítek tisíc kilometrů, neboť například vzdálenost mezi Prahou a New Yorkem je více než 6500km. Při takové vzdálenosti je jen navázání TCP spojení, které typicky obsahuje tři přenosy, jak ukazuje obrázek 3.3, zatížené zpožděním 31.85ms.



Obrázek 3.3: Schéma navázání TCP spojení

Tento problém je ještě umocněn přenosem malých souborů v rámci distribuovaného souborového systému. Typickým příkladem takovýchto přenosů jsou například domovské adresáře uživatelů na Unixových systémech. Už při prvním přihlášení jsou do domovského adresáře uživatele zapisovány malé soubory sloužící jako cache nebo konfigurace jednotlivých používaných služeb jako bash, Xorg a další. Domovský adresář uživatele je navíc nejčastěji čtenou složkou při provozování operačních systémů. Bylo by tedy žádoucí, aby k těmto datům byl co nejrychlejší přístup.

3.3 Možnosti řešení RTT zpoždění

Je zřejmé, že řešení negativních dopadů RTT zpoždění na přenosy v datové síti není na úrovni HW reálné. Technologie používané v geograficky oddělených lo-

kacích jsou velice nákladné, navíc vlastněné různými subjekty a jaká konkrétní technologie je na dané trase použita často nemáme možnost ani zjistit. Obdobná situace je i na národní úrovni, například v již zmiňované síti Cesnet. Možnosti jak vliv RTT v rámci distribuovaného souborového systému omezit ale existují a shodují se s řešeními pro provoz distribuovaného souborového systému na mobilním zařízení. V obou případech, byť z různých důvodů, chceme najít optimální datovou trasu a touto trasou přenášet maximální množství dat. Při hledání optimální cesty bude nutné brát v potaz nejen TTL a maximální kapacity linek, ale i RTT a hledat takovou trasu, kde bude TTL i RTT minimální za podmínky zachování maximální dostupné přenosové rychlosti. Na tento úkol by se na první pohled dal použít některý z již existujících dynamických směrovacích protokolů, které budou popsány v následující kapitole 4. Existující dynamické směrovací protokoly je však možné použít pouze v případě, že můžeme měnit hodnoty ve směrovačích mezi všemi uzly distribuovaného systému, což je typicky u geograficky rozsáhlých systémů nereálné. Řešením bude hledat optimální cestu na základě vlastností existujících dynamických směrovacích protokolů, ale hledání realizovat nad překryvnou sítí, kterou budou tvořit uzly distribuovaného systému, jak bude popsáno v kapitole 9.2.

4. Směrovací protokoly

Dnešní počítačové sítě jsou typicky stavěné tak, že mezi jednotlivými uzly existuje více paralelních datových cest. Tato násobná spojení zvyšují propustnost počítačové sítě, neboť provoz se může rozložit na více linek a zároveň zvyšují dostupnost, neboť v případě výpadku jedné, právě používané cesty, může být provoz odkloněn na alternativní cestu. Kterou cestou bude každý konkrétní paket odeslán se rozhoduje na základě směrovací tabulky, kterou obsahují jednotlivé směrovače, ale i jednotlivé servery i koncové stanice. Změnou hodnot směrovací tabulky můžeme ovlivnit směrování datových spojení, což jak bylo popsáno v předchozí podkapitole 3.3, je možná cesta k optimalizaci datových toků jak pro mobilní zařízení tak především pro geograficky rozsáhlé systémy a potažmo distribuované systémy. Směrovací tabulka obsahuje informaci kam je nutné poslat paket, aby dorazil k požadovanému cílu. Cílem může být konkrétní IP adresa nebo celá síť definované prefixem a maskou. Informace jsou do směrovací tabulky plněny trojím způsobem. Prvním způsobem vzniku je vložení záznamů o lokálně připojených sítích, druhým způsobem jsou staticky vložené záznamy administrátorem a třetím způsobem jsou záznamy vložené z dynamických směrovacích protokolů.

4.1 Lokálně připojené sítě

Základní hodnoty směrovací tabulky jsou plněny z lokálně přípojených sítí. Ze všech lokálních síťových karet jsou vyčteny informace o síti, do nichž patří IP adresy nastavené na těchto síťových kartách a ty jsou vloženy společně s maskou dané sítě do směrovací tabulky. V rámci směrovací tabulky mají tyto záznamy druhou nejvyšší prioritu, hned po statických záznamech, neboť se jedná o důvěryhodnou informaci, která vznikla na základě lokální dostupnosti dané sítě.

4.2 Statické záznamy

Druhým způsobem vzniku záznamů ve směrovací tabulce jsou staticky vložené záznamy, která administrátor ručně vloží do směrovací tabulky nebo nastaví jako součást konfigurace sítě pro dané zařízení. Typickým příkladem takovýchto záznamů je výchozí brána. Tyto manuálně vkládané záznamy mají ve směrovací tabulce nejvyšší prioritu.

4.3 Dynamické směrovací protokoly

Předchozí dva způsoby vzniku záznamů ve směrovací tabulce jsou nejběžnější a jsou oba v rámci konfigurace sítě libovolného zařízení použity vždy. Posledním zdrojem vzniku záznamů ve směrovací tabulce jsou dynamické směrovací protokoly. Důvody jejich vzniku jsou dva. Prvním je jednodušší údržba směrovacích tabulek pro velké sítě, kde jsou desítky až stovky směrovačů. V takovém případě by jakákoli ruční změna představovala velmi zdlouhavý proces, který by byl navíc díky ručnímu zásahu administrátora více náchylný k chybám. Druhým a pro tuto práci podstatnějším důvodem je možnost reagovat na změny v síti. Jak

bylo uvedeno v úvodu této kapitoly jsou dnes sítě stavěné za použití násobných datových cest a právě dynamicky směrovací protokol umožňuje efektivně těchto alternativních datových cest využít, neboť do směrovací tabulky dynamicky propaguje změny podle aktuálního stavu v síti. Podle toho na základě jakých údajů se vyhodnocují jednotlivé alternativní cesty rozdělujeme dynamické směrovací protokoly na distance-vector a link-state protokoly.

4.3.1 Distance-vector

Historicky starší a technicky jednodušší dynamické směrovací protokoly jsou protokoly označované jako distance-vector. U těch protokolů se jako metrika cesty sloužící k porovnání jednotlivých alternativních cest bere počet mezilehlých routerů mezi dvěmi sítěmi. Hlavními představiteli této skupiny směrovacích protokolů jsou RIP, RIP2 či RIPng[11]. Výhodou těchto protokolů je jednoduchost implementace a nenáročnost výpočtu metriky cesty. Zásadní problém je ale skutečnost, že tyto směrovací protokoly nijak nezohledňují výchozí parametry jednotlivých linek, jako je například maximální přenosová rychlost. Ještě problematičtější, především pro zamýšlené použití v této práci, je skutečnost, že tyto protokoly nereagují na jinou změnu stavu linky než je její úplný výpadek. Tento typ protokolů už se v praxi používá velmi zřídka a slouží často jen k výukovým účelům.

4.3.2 Link-state

Modernější a v aktuálně převážně používané protokoly jsou link-state protokoly. Protokoly patřící do této množiny, z nich nejpoužívanější je aktuálně OSPF[12], používají k nalezení nejlepší cesty kombinovanou metriku. Typicky je pro tuto metriku použita jako výchozí hodnota maximální rychlost daného spoje, což je parametr daný použitou technologií. K tomuto základnímu parametru se pak přidávají další, jako je zatížení linky, TTL či RTT. Metrika se počítá nejen při výpadku nějaké datové cesty nebo uzlu, ale opakovaně, což dovoluje rychleji zohledňovat měnící se parametry jednotlivých datových spojů. Navíc pro zrychlení konvergence protokolů této množiny je možné velké sítě v rámci dynamického směrovacího protokolu dělit na menší celky a jednotlivé metriky počítat samostatně pro menší počet uzlů, což výpočet výrazně urychlí. Jednotlivé oblasti, nazývané area, si pak na hraničních směrovačích vyměňují už jen sumarizované informace platné pro celou jednu oblast.

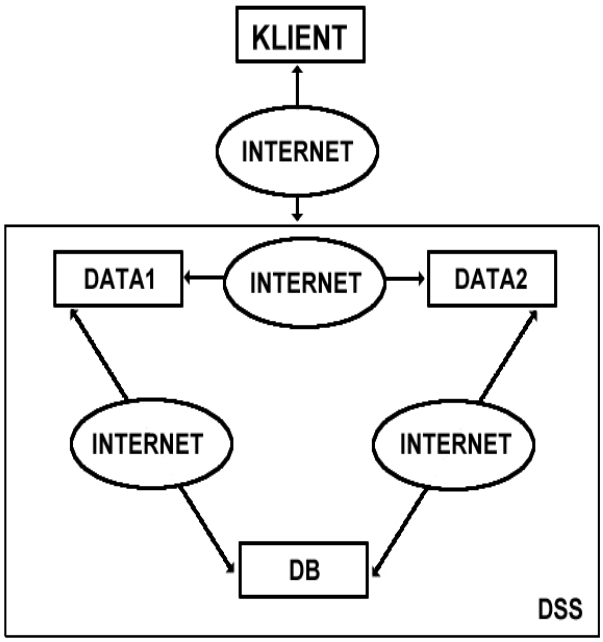
4.4 Dynamické směrovací protokoly v distribuovaném systému

Dynamické směrování by se dle předchozích odstavců ideálně hodilo jako podpůrný nástroj propojení distribuovaného souborového systému a mobilních sítí. Bohužel nasazení dynamických směrovacíh protokolů má v tomto modelu jedno zásadní omezení a tím je nutnost modifikovat směrovače v celé síti používané pro distribuovaný souborový systém a to včetně klientů. Tento požadavek je velmi omezující pro samotný distribuovaný souborový systém, neboť je typicky nasazován ve větších organizacích s více oddělenými pobočkami, které jsou spojené

linkami různých poskytovatelů. Použití na obecných mobilních sítích, stejně jako v geograficky rozsáhlých systémech je téměř vyloučené. Přestože není možné použít existující dynamické směrovací protokoly k propojení a optimalizaci datových cest v geograficky rozsáhlých distribuovaných souborových systémech, je možné použít obecné principy fungování dynamických směrovacích protokolů, především pak nejpoužívanějšího OSPF [13], v překryvné síti, jak bude popsáno v kapitole 9.

5. Distribuovaný souborový systém

Distribuovaný souborový systém je takový souborový systém, který nepracuje pouze s lokálními daty, ale umožňuje data sdílet prostřednictvím datové sítě. Uživateli je umožněno pracovat se souborovým systémem z libovolného místa v počítačové síti stejně, jako by se jednalo o souborový systém na pevném disku počítače uživatele. Základní model uspořádání distribuovaného souborového systému je znázorněn na obrázku 5.1. Distribuovaný souborový systém je typicky tvořen souborovými servery obsahujícími data uživatelů, na obrázku označenými jako DATA1 a DATA2, a databázovými servery obsahujícími metadata, na obrázku označenými jako DB. Tyto servery společně komunikují pomocí počítačové sítě, například internetu a z pohledu klienta tvoří jeden transparentní systém, ke kterému se klient opět prostřednictvím počítačové sítě připojuje. Jednotlivá data mohou být v distribuovaném souborovém systému replikována na více uzlech. Násobná existence dat je pro uživatele skryta a uživatel přistupuje k datům transparentně ať již jsou či nejsou replikována. Replikovaná data neslouží jen jako záloha v případě výpadku některého datového uzlu, ale zároveň mohou sloužit jako optimalizační nástroj, neboť požadavky mohou být směrovány podle aktuální zátěže jednotlivých uzlů či podle zatížení a dostupnosti jednotlivých datových linek.



Obrázek 5.1: Základní model distribuovaného souborového systému

Přínosy distribuovaného souborového systému jako je bezpečnost, škálovatelnost a odolnost vůči výpadkům sebou přinášejí negativa v podobě složitější implementace a vyšší režie než je tomu u běžného lokálního nebo síťového souborového systému. Přesto, že aktuálně existuje celá řada distribuovaných souborových systémů, většina nedisponuje všemi stěžejními vlastnostmi popisovanými v

následujících podkapitolách. Často se jedná o částečné zobecnění systémů, které jsou vyvíjeny pro jeden konkrétní cíl. Současná existující řešení a jejich vlastnosti budou popsány na konci této kapitoly.

5.1 Základní charakteristika

Souborové systémy jsou implementovány nad virtuálním souborovým systémem obsaženým v jádře operačního systému. Virtuální souborový systém je součástí jádra operačního systému a poskytuje zobecněné a jednotné rozhraní jednotlivým konkrétním souborovým systémům. Všechny operace nad souborovým systémem jsou díky virtuálnímu souborovému systému volány jednotně. Virtuální souborový systém slouží jako překryvná vrstva, která předává volání jednotlivých operací na konkrétní modul jádra operačního systému podle aktuálně používaného souborové systému, kterým může být jak lokální souborový systém jako například EXT2[14], XFS[15] či ReiserFS[16], tak distribuovaný souborový systém jako je Coda[56], OpenAFS[60] nebo NFS[50]. Samotnou operaci nad souborovým systémem vykonává až konkrétní modul jádra operačního systému. Výhodou tohoto modelu implementace je, že aplikace přistupují ke všem souborovým systémům stejně.

Distribuovaný souborový systém může být implementován jako modul jádra operačního systému nebo jako externí program v uživatelském režimu s podporou v jádře operačního systému. Obě tyto varianty mají své výhody i nevýhody. Distribuovaný souborový systém implementovaný v jádře operačního systému je v provozu rychlejší, neboť dochází k menšímu množství předávání dat mezi aplikacemi a jádrem operačního systému. Aplikace požádá jádro operačního systému o operaci se souborovým systémem a to jej přímo vyřídí. Na druhou stranu, pokud je distribuovaný souborový systém implementován v jádře, může mít chyba v něm či případné prolomení bezpečnostní ochrany zásadní negativní následky nejen pro data uložená v distribuovaném souborovém systému, ale pro celý operační systém, který je tak také potenciálně ohrožen. Naopak, pokud je distribuovaný souborový systém implementován v uživatelském módu, je zajištěna větší bezpečnost a stabilita systému, avšak veškeré požadavky musí aplikace posílat prostřednictvím jádra operačního systému, což provoz zpomaluje. Další zpomalení způsobuje fakt, že všechna data, která jsou přenášena od aplikace nebo k ní, musí být kopírována přes jádro operačního systému. Možnost komunikace přes jádro operačního systému může být zajištěna samostatným modulem jádra operačního systému, jako je tomu například u OpenAFS, které bude detailněji představeno v kapitole 5.9.4, nebo pomocí univerzálního modulu Fuse[17], které slouží jako obecný komunikační interface při implementaci souborových systémů v uživatelském módu operačního systému. Příkladem implementace souborového systému pomocí Fuse je například linuxová implementace masFS[25] nebo EncFS[26]. Jak bude popsáno v kapitole 5.9, aktuálně nejpoužívanějším modelem implementace je varianta se samostatným modulem jádra operačního systému, který komunikuje se serverovou částí implementovanou v rámci uživatelského módu operačního systému. Důvodem volby této implementace je výrazně jednodušší ladění celého programu, neboť drtivá většina systémů je implementována v rámci uživatelského módu a jen nejnutnější části jsou implementovány v modulu jádra operačního systému.

5.2 Komunikace

Distribuovaný souborový systém ke svému chodu využívá datovou síť. Obecně může být pro distribuovaný souborový systém použit libovolný protokol jako TCP/IP[18], IPX[20], NetBIOS[21], UDP[19]. Konkrétní použití závisí na konkrétní implementaci daného distribuovaného souborového systému. Každý z protokolů má své klady i zápory a podle toho jsou i používány. Například IPX není používán mimo místní sítě, neboť neumožňuje směrování [22]. Nejvíce jsou ve světě distribuovaných souborových systémů používány protokoly rodiny TCP/IP a to jak TCP tak UDP. Každý má své pro a proti. TCP doručuje pakety spolehlivě, v pořadí v jakém byly odeslány a pokud některé pakety nejdou nebo dojdou poškozeny, TCP protokol zajistí jejich znovu zaslání. Tato spolehlivost doručování je zaplacena pomalejším navazováním spojení a pomalejší komunikací, neboť jednotlivé přenosy dat je nutné vždy potvrzovat. Tyto nevýhody se výrazněji projevují na více zatížených či nespolehlivých linkách. Naproti tomu UDP protokol disponuje úspornější režií přenosu, neboť přenos pomocí UDP protokolu není nijak potvrzovaný a chyby v přenosu či v navazání komunikace nejsou nijak řešeny a jejich řešení je přenecháno vyšším vrstvám ISO/OSI modulu či aplikaci samotné.

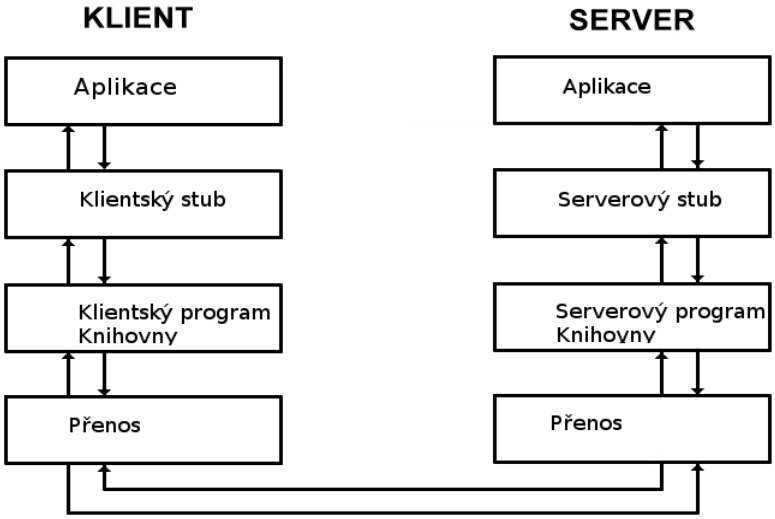
Klient a jednotlivé uzly distribuovaného souborového systému komunikují na jednom či více portech a to staticky nastavených či dynamicky zvolených. Ideální je, aby distribuovaný souborový systém komunikoval pouze na jednom či několika pevně stanovených portech. Pokud jsou porty voleny v rámci systému dynamicky, vzniká problém při přístupu k datům přes firewally, neboť pro dynamicky přidělované porty se velice složitě definují vhodná firewallová pravidla, která by jednak umožnila přístup na patřičnou službu pouze z definovaných míst v síti a zároveň spolehlivě ochránila síť za firewallem. Typickou ukázkou systému, kde tento problém vzniká je NFS verze 2 a 3, která používá pro svoji komunikaci RPC protokol[27]. Komunikace pomocí RPC je realizována na několika dynamicky volených portech, proto je také NFS jen velice zřídka používán volně v prostředí internetu, ale využívá se spíše v uzavřených lokálních sítích.

Distribuovaný souborový systém je tvořen množinou decentralizovaných serverů, které společně tvoří jeden ucelený systém. Pro klienta je přístup k datům transparentní, ale jednotlivá data jsou fyzicky rozmístěna na jednotlivé uzly, které spolu vzájemně komunikují. Komunikace může být realizována v několika modelech jako jsou vzdálené volání, zasílání zpráv, komunikace pomocí front zpráv či stream dat.

Vzdálené volání procedur

Z hlediska implementace je nejjednodušší možností komunikace distribuovaných systémů vzdálené volaní procedur RPC či pro objektové jazyky vzdálené volání metod RMI[28]. Volání je v kódu realizováno stejně jako volání klasické procedury nebo metody. Volaná procedura nebo metoda je zpracována knihovní vrstvou RPC/RMI, která lokální volání přenese na vzdálený počítač, kde je požadovaná operace provedena. Tento model má svá omezení, kterým je především odkaz na lokální data nebo I/O zařízení, která jsou na lokálním a vzdáleném uzlu jiná. Tento problém řeší speciální adaptér nazývaný Stub. Tento adaptér je klientský a serverový. Klientská část připraví vše potřebné pro zavolání metody či pro-

cedury na vzdáleném uzlu a přibalí k požadavku volání nutná data, pak vyhledá serverový adaptér na vzdáleném uzlu. Tam se požadavek přenese a serverový adaptér předaná data připraví ke zpracování a obrácenou cestou zašle zpět odpověď. Model této komunikace je znázorněn na obrázku 5.2. Tento komunikační



Obrázek 5.2: RPC - vzdálené volání procedur

model je jednoduchý na implementaci pro programátora, ale je drahý z hlediska režie přenosů, neboť pro každý požadavek je třeba sestavovat a realizovat samostatné spojení. Navíc je pro komunikaci RPC využíváno dynamické mapování portů pomocí programů Portmap[29] nebo RPCBind[30], které není jednoduché provozovat za firewallem, neboť používají různé pevně nedefinované porty.

Zasílání zpráv

Druhou možností komunikace v distribuovaném systému je zasílání zpráv. Při zasílání zpráv nevybírá inicializátor komunikace proceduru nebo metodu, kterou chce spustit, ale pouze předá zprávu, která musí odpovídat vzájemně dohodnutému formátu a příjemce zprávy zprávu zpracuje. Implementací komunikace pomocí zpráv je více, například Berkley sockety[23] nebo Message-passing interface MPI[31]. Zasílání zpráv je dnes nejčastěji realizováno pomocí socketů. Socket je programový komunikační nástroj, který je schopen přenášet data mezi dvěma nebo více účastníky komunikace. Velká výhoda socketů je v tom, že se jedná o univerzální komunikační interface, který je schopen používat různé přenosové protokoly. Nejčastěji bývají používány protokoly TCP nebo UDP. Komunikace socketů je adresována pomocí typu socketu a adresy komunikujícího uzlu. Adresa se liší podle použitého typu socketu. Pro AF_INET sockety je adresou IP adresa a port uzlu se kterým chceme komunikovat, ale například pro AF_UNIX sockety je touto adresou cesta k socketu v rámci lokálního souborového systému. Díky tomuto způsobu adresace je snadné pro takový styl komunikace nakonfigurovat firewallová pravidla, což je v distribuovaném prostředí s ohledem na bezpečnost

důležitá vlastnost. Další pozitivní vlastností socketů je jejich obecnost, kde je možné s minimem úsilí vyměnit v kódu jeden typ socketů za jiný. Pak realizace spojení v rámci kódu probíhá velmi podobně, ale vlastnosti použitých socketů se mohou výrazně lišit. Příkladem je například výměna AF_INET sockety za AF_UNIX. AF_UNIX socket může být použit pouze v rámci jednoho uzlu, ale režie realizace spojení je u něj výrazně nižší než u AF_INET socketů. Těchto vlastností se dá s výhodou využít při interní komunikaci v rámci jediného uzlu, jak bude popsáno v kapitole 7.2.1. V současnosti se jedná o nejpoužívanější formu komunikace v distribuovaných systémech.

Stream

Třetí možností komunikace jsou datové streamy. Komunikace typu stream se používá v situacích, kdy je jedním spojením třeba přenést velké množství dat mezi dvěma pevně danými uzly a to s minimálním zpožděním. Typickým příkladem takovéto komunikace je přenos videa či hlasu, kde není zásadní velice rychlé opakované navázání spojení, ale až už se spojení naváže je třeba jím přenášet data s co nejmenší režií. Pro přenos hlasu či videa je tato komunikace navíc atypická tím, že se nemusí nutně jednat o zcela spolehlivou komunikaci, neboť při hlasovém či video streamu není malé množství ztracených dat poznat. Naopak je tento styl komunikaci citlivý na synchronizaci, tedy aby nedocházelo ke zpožďování přenosu. Dalším specifikem streamové komunikace je skutečnost, že se často jedná o komunikaci s přepínáním kanálů a nikoliv zpráv. Tedy na začátku komunikace mezi dvěma uzly se nalezne cesta a ta je pak používána po celou dobu spojení či existence cesty. Pozitivní důsledek tohoto chování je minimalizace režie přenosu, neboť není nutné cestu hledat pro každou jednotlivou zprávu. Příkladem komunikace, kde dochází k přepínání kanálů je například protokol ATM[32]. Přestože minimální režie je v distribuovaných soubových systémech důležitým požadavkem, tento typ komunikace, díky své režii na navázání spojení a především s ohledem na možnost ztráty části zpráv či jen velkému zpoždění danému použitím v aktuálním nevýhodném kanálu, není používán ke komunikace uvnitř distribuovaného souborového systému.

5.3 Replikace

Replikace je proces tvorby násobných kopií dat na různých uzlech distribuovaného systému. Tato možnost duplikace dat v rámci distribuovaných systémů přináší řadu pozitivních vlastností, ale i podstatné komplikace. Základní výhodou použití replikací v distribuovaných souborových systémech je zvýšení dostupnosti a bezpečnosti dat. Data mohou být umístěna na různých místech sítě, které od sebe mohou být geograficky vzdálené. Rozmístění uzlů systému do vzájemně vzdálených míst, která se vzájemně neovlivňují, tvoří geografický cluster, který v případě výpadku určité části systému, tedy jednotlivého uzlu či celé lokality, umožní systém dále používat tak, aniž by běžný uživatel zaznamenal výpadek. Druhou výhodou využití replikace je možnost load balancingu, tedy rozkladu zátěže systému. Při použití replikací je možné náhodně či cíleně směrovat dotazy na jednotlivé replikované uzly a tím odlehčit více zatíženým částem systému a zvýšit tak průchodnost celého distribuovaného systému. Podstatné je, že existence repli-

kovaných dat je pro uživatele transparentní. Uživatel nepozná a ani nepotřebuje poznat, zda pracuje s jedinečnou kopií dat nebo s daty, která jsou replikovaná. Hlavním problémem při použití replikací je nutnost zajistit konzistentní stav dat v systému, tedy zajistit stav, kdy při stejném dotazu na jakýkoliv uzel systému dostaneme stejnou odpověď. Tento požadavek je pro některé specifické provozy, jako jsou například systémy CDN[33], příliš silný. V systémech jako je CDN jsou sice požadována aktuální data, ale větší důraz je kladen na dostupnost dat a krátkodobá neaktuálnost je tolerována. Běžně se takové služby používají na odbavení velkého množství požadavků na málo se měnící data, jako jsou například kaskádové styly či grafika webových stránek.

Pro distribuovaný souborový systém také není nutně potřebné, aby každý uzel měl okamžitě aktuální data. Ale požadujeme, aby každý realizovaný požadavek do systému vrátil aktuální data. Data, která budou v systému změněna, se postupně aktualizují v jednotlivých uzlech, ale uživateli bude vždy předána aktuální odpověd.

Ať už požadujeme data úplně aktuální či ne, základní nutný požadavek je, aby se v rámci systému nemohla na různých místech měnit ta samá data. Tedy, aby neproběhl současně různý zápis do stejných dat na různých replikách, neboť v tom případě je problém určit, která data jsou správná. Možností řešení je několik.

Základní možností je Master-Write replikace. Jedna replika v rámci systému je zapisovatelná. Všechny ostatní repliky slouží pouze ke čtení. Uživatel v rámci systému dostává dočasně právo zápisu do dané kopie. Pokud jej nevyužije v daném čase, je právo předáno dalšímu zájemci a původní vlastník oprávnění je při pokusu o uložení dat upozorněn, že soubor byl modifikován. Tento postup je implementačně snadný, neboť zápis může proběhnout jen na jediné kopii dat. Díky tomu, že se pro Master-Write kopii řeší souběžný zápis na centralizovaném systému, lze k jeho řešení použít například jen zámky na soubory. Samozřejmě platnost zámku musí být časově omezena, aby nedocházelo k trvalým uzamčením souborů, například po neočekávaném odpojení klienta. Tím je zajištěno, že si uživatelé nemohou vzájemně přepisovat stejné data. Může se stát, že zapisovatelná kopie nebude nějaký čas dostupná. Pak je možné z nějaké repliky určené do té doby pouze pro čtení vytvořit zapisovatelnou repliku. Tato informace se musí distribuovat po všech uzlech nebo v rámci databáze, která dané informace spravuje.

Druhou možností zajištění konzistence změn v distribuovaném systému je řazení požadavků podle času vzniku. Čas nemusí být v tomto případě ani reálným fyzikálním údajem, ale jen možností jak zajistit posloupnost operací. Pak je možné říci, že aktuální je vždy poslední změna, což je možné pomocí času a transakcí v systému zajistit. Tato metoda je výhodnější z pohledu odolnosti vůči výpadku, neboť neexistuje žádná vyhrazená replika pro zápis, ale distribuovaný algoritmus, kterým uzly určují pořadí operací. Tato obecnost se však negativně projevuje na počtu synchronizačních požadavků v distribuovaném systému.

5.4 Konzistence

Vzhledem k tomu, že se jedná o distribuovaný systém, navíc s podporou replikací, je nutné řešit konzistenci dat. Data distribuovaného systému jsou konzistentní, pokud je s nimi v celém systému pracováno dle dohodnutého modelu. V distribu-

ovaných systémech definujeme několik modelů konzistence. Konzistentní model je dohoda či algoritmus, kterým se řídí všechny procesy přistupující k datům. Jednotlivé modely konzistence disponují různou sílou konzistence. Sílou konzistence se rozumí míra možné odchylky od jednotného stavu dat. Čím silnější je konzistence daného modelu, tím jednotnější pohled na data zajišťuje, ale také tím více zpráv se v rámci systému musí zaslat k docílení požadovaného stavu dat.

5.4.1 Konzistence v centralizovaných systémech

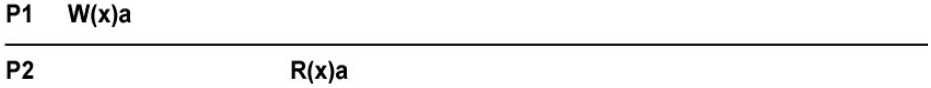
Problém konzistence je jednoduše řešitelný v rámci centralizovaných systémů, kde jsou data uložena na jednom serveru. Přistupující procesy a jejich požadavky jsou seřazeny sekvenčně podle času, neboť čas je jediný a společný na serveru a jednotlivé požadavky jsou vykonávány sekvenčně, tak jak na server dorazily. Zajištění dat je v centralizovaných systémech snadno realizovatelné pomocí zámků ať už na souborovém systému nebo ve sdílené paměti procesů přistupujících k datům.

5.4.2 Konzistence v distribuovaných systémech

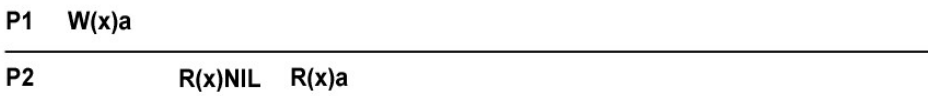
Složitější problém nastává u distribuovaných systémů, neboť distribuovaný systém nedisponuje jediným časem ani sdílenou pamětí či souborovým systémem a požadavky na operace s daty mohou vznikat na více uzlech současně. Každý uzel udržuje svůj čas, který se i přes cyklickou synchronizaci postupně rozchází s časem ostatních uzlů. Také je třeba uvažovat zpoždění zpráv přenášených sítí. Navíc v distribuovaném systému neexistuje jedno místo, kde je možné o pořadí požadavků rozhodnout, ale je třeba použít distribuovaný algoritmus, například hlasovat jak je detailněji popsáno v kapitole 5.5.2. Tím, že v distribuovaných systémech nemáme jediný synchronizační bod, nemáme ani společný globální stav systému, ale na všechny operace je třeba nahlížet jako na paralelně běžící procesy. V následujících kapitolách si představíme nejběžnější používané modely konzistence.

5.4.3 Striktní konzistence

Striktní model konzistence předpokládá, že jakékoliv čtení z datového bloku X, vždy vrátí hodnotu posledního zápisu do bloku X. Tento model nativně existuje v centralizovaných jednoprocesorových systémech, neboť předpokládá existenci jednotného času a jednoho synchronizačního místa. V distribuovaných systémech je nedosažitelný. Volání procesů dodržujících sktriktní konzistenci je znázorněno na obrázku 5.3. Na obrázku 5.4 je znázorněno volání procesů porušujících striktní konzistenci. Striktní konzistence zavádí absolutní časové uspořádání sdílených přístupů.



Obrázek 5.3: Procesy dodržující striktní konzistenci



Obrázek 5.4: Procesy nedodržující skritní konzistenci

5.4.4 Sekvenční konzistence

Při použití sekvenční konzistence se snažíme dosáhnout stavu, kde jednotlivé operace budou provedeny ve všech procesech ve stejném pořadí. Provedení operací v jednotlivých procesech není realizováno synchronně, mezi jejich provedením může docházet ke zpoždění. Tento model konzistence je slabší než model striktní konzistence, ale jednoduše implementovatelný. Jedním z příkladů implementace může být například implementace pomocí časových značek a protokolu ABCAST[34], který se stará o distribuci synchronizačních zpráv pomocí kterých se hlasuje o tvorbě časových značek. Abcast protokol funguje tak, že proces, který požaduje vykonání operace vyšle všem ostatním členům skupiny žádost o provedení operace. Všichni členové jsou povinni odpovědět svým návrhem časové značky. Nejvyšší časová značka bude použita. Jednotliví členové skupiny tak vytvářejí lokální fronty sekvenčně seřazených operací. Příklad sekvenční konzistence je znázorněn na obrázku 5.5 a na obrázku 5.6 jsou znázorněny procesy porušující pravidla sekvenční konzistence.

Obrázek 5.5: Procesy dodržující sekvenční konzistenci

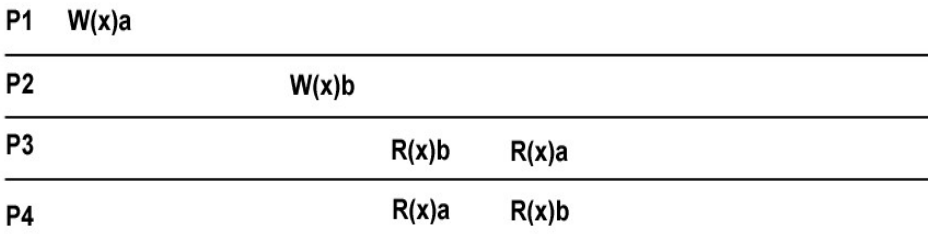
Obrázek 5.6: Procesy nedodržující sekvenční konzistenci

5.4.5 Kauzální konzistence

Při kauzální konzistenci rozlišujeme procesy, které jsou nezávislé a procesy, které jsou potenciálně kauzálně závislé. V kauzální konzistenci musí být operace potenciálně kauzálně závislé viděny ve všech procesech ve stejném pořadí. Operace, které nejsou na jiných závislé, mohou být v různých procesech viděny v různém pořadí. Pro implementaci kauzální konzistence je nutné zavést a udržovat graf závislosti operací. Model kauzální konzistence může být implementován například pomocí logických hodin 5.5.2. Příklad kauzální konzistence na čtyřech kauzálně nevázaných procesech je znázorněno na obrázku 5.7.

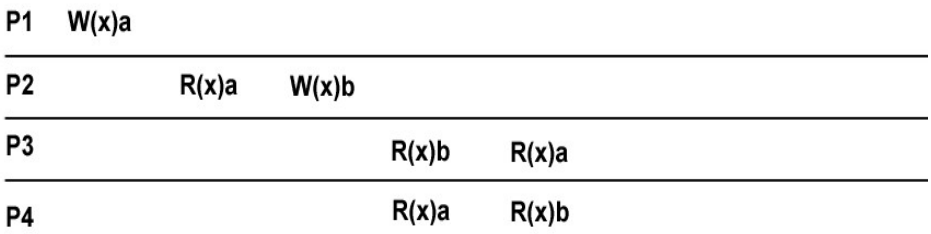
Obrázek 5.7: Kauzální model konzistence s kauzálně nezávislými procesy

Na obrázku 5.8 jsou znázorněny kauzálně závislé procesy, které splňují parametry kauzální konzistence pro kauzálně vázaně procesy.



Obrázek 5.8: Kauzálně závislé procesy dodržující kauzální konzistenci.

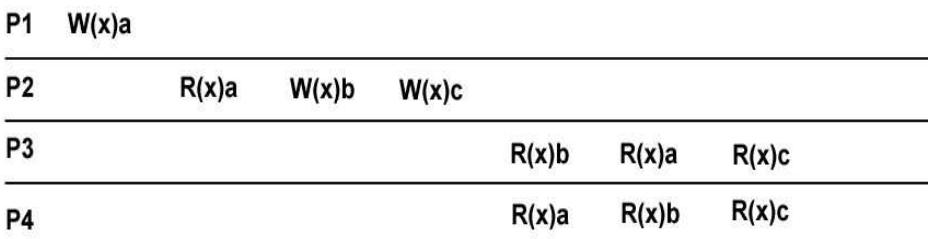
Na obrázku 5.9 jsou znázorněny kauzálně závislé procesy, které nesplňují pravidla kauzální konzistence.



Obrázek 5.9: Kauzálně závislé procesy nedodržující kauzální konzistenci.

5.4.6 FIFO konzistence

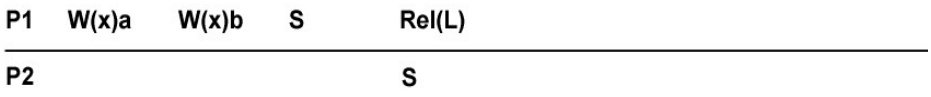
V FIFO konzistenci požadujeme, aby operace prováděné v konkrétním pořadí jedním procesem, byly ostatními procesy viděny v tom samém pořadí jako jsou prováděny v prvním procesu. Naopak zápisy prováděné různými procesy mohou být v dalších procesech viděny v různém pořadí. Příklad procesů pro FIFO konzistenci je znázorněn na obrázku 5.10.



Obrázek 5.10: Procesy fungující s FIFO konzistencí

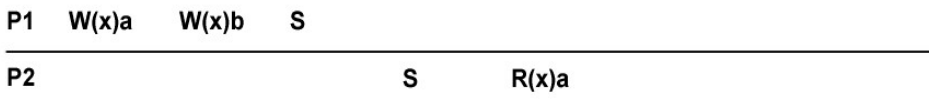
5.4.7 Slabá konzistence

Při použití slabé konzistence je nutné zavést synchronizační proměnné - *S*. K těmto synchronizačním proměnným se přistupuje podle pravidel sekvenční konzistence, které už byly popsány v předchozích kapitolách. Sekvenční řazení operací je možné realizovat například použitím logických hodin. Žádné operace se synchronizačními proměnnými není možné provést dříve, než skončí předchozí operace nad synchronizační proměnou. Jednotlivé změny jsou prováděny lokálně a jen výsledný stav při uvolnění synchronizační proměnné je zveřejněm jako globální stav dané proměnné. Pro přístup k synchronizačním proměnným je možné použít například Lamportův algoritmus pro vzájemné vyloučení [35], který ošetřuje přístup více procesů ke kritické sekci. Sdílená data jsou konzistentní pouze po synchronizaci. Příklad operací splňujících pravidla slabé konzistence jsou znázorněna na obrázku 5.11.



Obrázek 5.11: Procesy splňující pravidla slabé konzistence

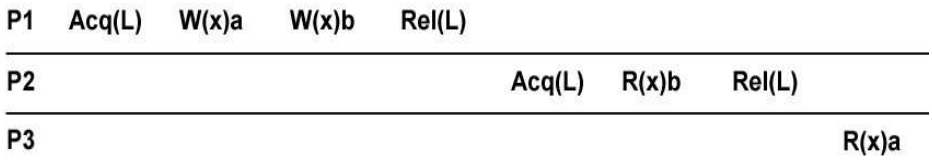
Na obrázku 5.12 jsou znázorněny procesy nedodržující pravidla slabé konzistence.



Obrázek 5.12: Procesy nesplňující pravidla slabé konzistence

5.4.8 Uvolňovací konzistence

Pro implementace uvolňovací konzistence se zavádí operace požadavku přístupu - ACK a následně uvolnění přístupu - REL. Před provedením operace čtení nebo zápisu musí být dokončena operace požadavku na přístup ACK. Před provedením operace uvolnění REL musejí být dokončeny všechny operace čtení a zápisu. Jednotlivé operace ACK a REL musí být FIFO konzistentní. Podobně jako s operacemi P() a V() u semaforu, zde realizujeme ošetření kritické sekce. Data jsou konzistentní až po opuštění kritické sekce. Posloupnost operací splňující uvolňovací posloupnost je znázorněna na obrázku 5.13 .



Obrázek 5.13: Procesy splňující pravidla uvolňovací konzistence

5.4.9 Přístupová konzistence

Stejně jako u uvolňující konzistence i pro přístupovou konzistenci se zavádí operace pro přístup ACK a uvolnění REL. Přístup procesů k datům není povolen, pokud nebyly dokončeny všechny aktualizace chráněných dat daného procesu. Přístup pro zápis je procesu umožněn pouze v případě, že žádný jiný proces nepožaduje operaci zápisu ani čtení nad chráněnými daty. Po operaci zápis si následný proces čtení musí vyžádat aktuální kopii dat od aktuálního vlastníka dat. Sdílená data jsou konzistentní vždy při vstupu do kritické sekce. Každý blok chráněných dat má definovanou svoji kritickou sekci. Posloupnost operací splňující pravidla přístupové konzistence jsou znázorněna na obrázku 5.14 .

5.5 Hodiny a synchronizace času

Pro zavedení konzistentních modelů, potřebujeme hodnotu, podle které budeme moci jednotlivé požadavky řadit. V centralizovaných systémech s jediným uzlem je situace jednoduchá, neboť můžeme využít reálný čas, který je kontrolován

P1	Acq(Lx)	W(x)a	Acq(Ly)	W(y)b	Rel(Lx)	Rel(Ly)		
P2						Acq(Lx)	R(x)a	R(y)NIL
P3							Acq(Ly)	R(y)b

Obrázek 5.14: Procesy splňující pravidla přístupové konzistence

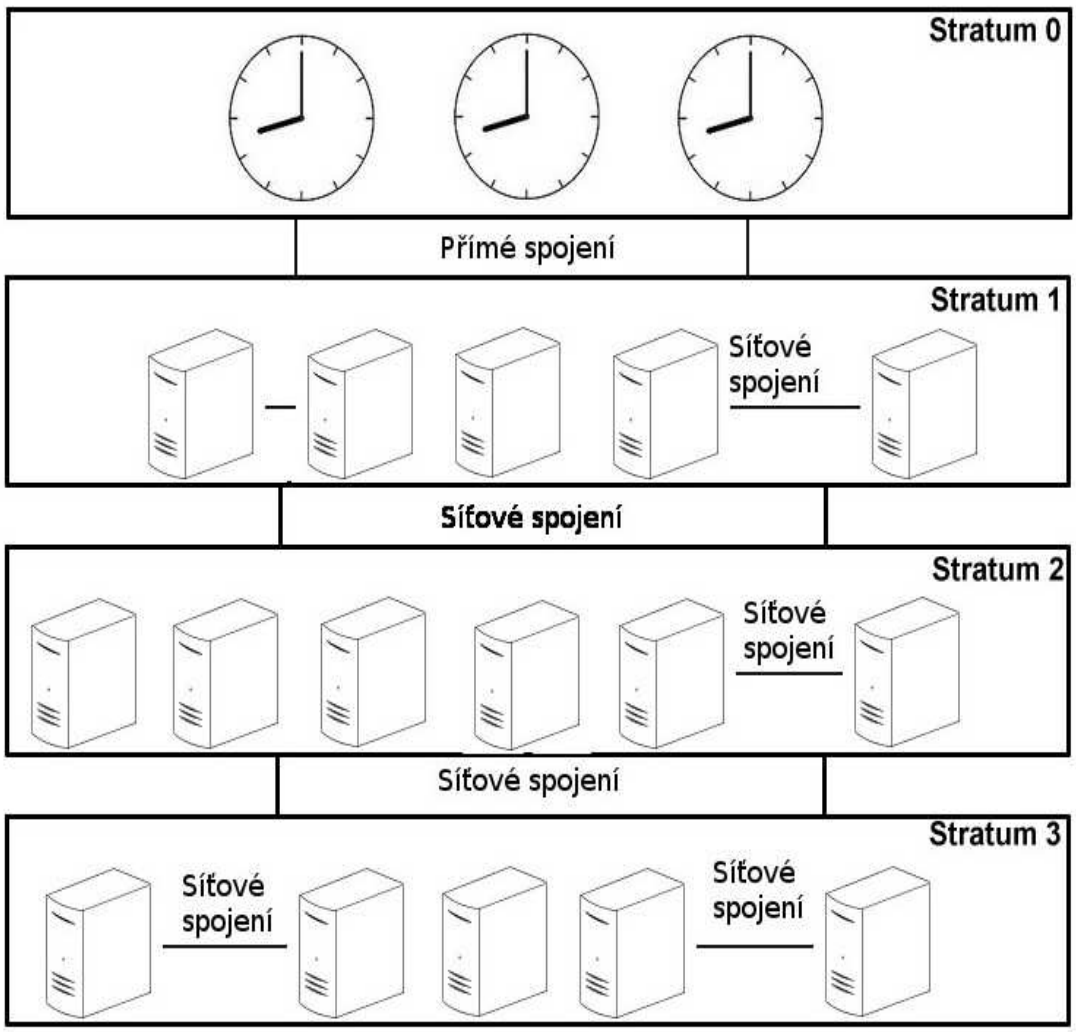
pouze v rámci jediného uzlu. V distribuovaných systémech, které mají více než jeden uzel a ještě často geograficky oddělené či dokonce implementované pomocí různých technologií, není čas pouze jeden, ale vlastní časové údaje má každý uzel. Tento lokální čas, pokud není dostatečně často aktualizován se postupně desynchronizuje vůči ostatním uzlům. Další problém vzniká při zasílání požadavků pomocí počítačové sítě. Při tomto přenosu dochází k různě dlouhým a nepredikovatelným prodlevám. Vzhledem k těmto skutečnostem není jednoduché určit přesné pořadí jednotlivých operací, které je nutné pro synchronizaci replikací a tedy zajištění požadované úrovně konzistence.

Jelikož použití hodin v distribuovaných systémech nesouvisí přímo s časem jakožko fyzikální veličinou, ale cílem jeho použití je pouze stanovit jednotné pořadí operací prováděných na jednotlivých uzlech distribuovaného souborového systému, používají se pro tento účel logické hodiny, které budou popsané v kapitole 5.5.2.

5.5.1 Fyzické hodiny

Při použití fyzických hodin jakožto měrné veličiny pro určení pořadí operací v distribuovaných systémech pracujeme s reálným časem. Jednotlivé operace jsou řazené podle časové posloupnosti svého vzniku. Pro toto použití je nutný velice přesný čas, aby bylo možné přesně rozlišit pořadí vzniku jednotlivých operací, které mohou vznikat na různých uzlech distribuovaného systému ve velmi těsném časovém sledu. Implementace řazení operací je nejjednodušší, ale také nejméně přesná. Důvodem je desynchronizace hodin na jednolivých uzlech a problém zpoždění síťových přenosů při opětovné synchronizaci hodin. Existuje několik používaných algoritmů, které se snaží zajistit co nejrychlejší a nejpřesnější synchronizaci reálného času v distribuovaných systémech. Nejpoužívanější principy synchronizace času budou popsány v následujících kapitolách. Nejběžněji používaná metoda synchronizace fyzického času v distribuovaných systémech je implemetována v NTP protokolu[99], který je popsán v *RFC 778, 891, 956, 958 a 1305*. Aktuální verze NTP je verze 4, kterou popisuje *RFC 5905*. Princip vychází z distribuovaného algoritmu. Všechny uzly systému jsou napojeny na společný NTP server, který jednotlivým uzlům umožní synchronizovat svůj lokální čas. NTP servery využívají hierarchický model, který je znázorněn na obrázku 5.15.

Jednotlivé úrovně se nazývají stratum a jsou označeny číslem 0 až 15, které označuje kvalitu zdroje času. Čím nižší číslo, tím přesnější hodnota. Servery stratum 15 jsou jen teoretické, v reálu se vyskytují hodnoty do úrovně stratum 5. Jednotlivé úrovně modelu čerpají čas z nadřazeného zdroje, jak ukazuje následující tabulka.



Obrázek 5.15: Hierarchické uspořádání NTP serverů

Číselná hodnota použitá pro synchronizaci času má délku 64 bitů a využívá pevnou desetinnou čárku. Prvních 32 bitů slouží k přenosu počtu vteřin od 1.1.1900, druhých 32 tvoří desetinou část. Během synchronizace času klient nejprve od všech dostupných NTP serverů zjistí jejich aktuální čas, stratum a přesnost serveru. Ze získaných hodnot vyřadí ty, které se svojí hodnotou aktuálního času výrazně odlišují od ostatních. Ze zbylých přijatých hodnot vypočítá svůj nový čas a pomocí klouzavého algoritmu přibližuje své interní hodiny k této hodnotě. Pokud je odchylka časů větší než 128ms, nepoužije se postupná korekce klouzavým algoritmem, protože přibližování by bylo příliš pomalé, ale mění se čas skokově. V případě, že je odchylka větší než 17 minut, změna času se neprovede vůbec, ale je ohlášena chyba. I zde nastává problém. Mezi odesláním NTP požadavku na provedení synchronizace času, doručením odpovědi od NTP serveru a následnou korekcí času vzniká nenulová časová prodleva. Tato prodleva je o to delší čím větší je vzdálenost mezi NTP serverem a uzlem žádajícím synchronizaci a čím více zatížená či méně spolehlivá je použitá datová síť. Reálně tedy dojde k další desynchronizaci už během vyřizování synchronizačních požadavků. V geograficky méně rozsáhlých a na počet klientů a operací méně vytížených systémech můžeme teoreticky toto zpoždění zanedbávat, ale systém musí být schopen pracovat s

Stratum	Zdroj času
stratum 0	Obsahují přesné atomové hodiny, nejsou přímo dostupné ze sítě, ale jen ze serveru stratum 1
stratum 1	Čerpají čas po interních sítích ze serveru stratum 0, běžně označované jako time servery
stratum 2	Čerpají čas ze serverů stratum 1
stratum 3-14	Další servery čerpající čas vždy z vyšší úrovně
stratum 15	Servery, který už neposkytují přesný čas, nedošlo u nich k synchronizaci, například v důsledku výpadku sítě

Tabulka 5.1: Zdroje času pro jednotlivé úrovně NTP serverů.

určitou časovou tolerancí. Příkladem takového systému je například Kerberos[36], který používá i KIVFS, jak bude uvedeno v následujících kapitolách.

5.5.2 Logické hodiny

Pro fungování distribuovaného souborového systému, ve kterém na velkém množství uzlů vzniká velké množství požadavků, které je třeba řadit jsou fyzické hodiny s ohledem na předchozí kapitolu velice problematicky použitelné, neboť reálně nikdy nenastane stav plné synchronizace hodin na všech uzlech po delší časový úsek. Avšak pro chod distribuovaného systému je zásadní určení jednoznačného pořadí operací, které nemusí být na fyzický čas vázané. Na základě tohoto principu se začali používat logické hodiny. V případě logických hodin se nejedná o časový údaj, ale pouze o kladné číslo určující pořadí dané operace v rámci celého systému. Pro centralizované systémy je určení jednoznačného pořadí jednoduché například pomocí ošetření kritické sekce, v rámci které se očíslují jednotlivé operaci. Díky operacím P() a V() nad kritickou sekcí máme zaručeno, že do kritické sekce, tedy místa přidělení logických hodin, se dostane vždy jen jeden proces. V distribuovaných systémech není jednoduše možné tyto operace P() a V(), které vyžadují sdílenou paměť, použít. V distribuovaných systémech si každý uzel udržuje své lokální logické hodiny, které představují počítadlo operací. Tento údaj se pomocí zasílání synchronizačních zpráv a dodržování dohodnutého algoritmu transformuje do globálního času, tedy globálních logických hodin, podle kterých budou na všech uzlech jednotlivé operace vykonány. Zásadní pro správnou funkčnost je dodržování stejného algoritmu tvorby logických hodin, neboť možných implementací existuje více. V rámci tvorby logických hodin se vytvářejí relace mezi jednotlivými událostmi. Tyto relace jsou transitivní. Pokud máme operaci A a operaci B, kde operace A předchází operaci B a zároveň máme operaci C, jenž předchází operaci B, můžeme říci, že operace A předchází operaci C. Každá implementace logických hodin obsahuje dvě pravidla:

- R1: toto pravidlo určuje jakým způsobem jsou logické hodiny synchronizovány při provádění operace s daty (odeslání / příjem)
- R2: toto pravidlo určuje princip synchronizace logických hodin z globálního pohledu

Skalární Lamportovy hodiny

Nejjednodušší formou logických hodin jsou skalární nebo také Lamportovi hodiny [40], které jsou realizované pomocí jediného čísla - počítadla pro každý jednotlivý uzel v systému. Číslování je prováděno pomocí sekvenční inkrementace lokálního počítadla operací. Při vzájemné komunikaci si jednotlivé servery synchronizují své lokální logické hodiny tím, že si vzájemně vymění informace o stavu svého lokálního počítadla a společně použijí pro další operaci nejvyšší hodnotu. Lamportova definice skalárních hodin z roku 1978 definuje pro každý proces P_i vlastní lokální hodiny C_i, které přiřazují každé lokální události a číslo $C_i(a)$. Dále definuje globální skalární hodiny C. Pro události a a b z předchozího odstavce platí, že událost a předchází událost b, pak pro jejich skalární hodiny platí, že $C(a) < C(b)$. Pro skalární Lamportovi hodiny platí pravidla R_1 a R_2:

- R_1: před provedením každé jednotlivé operace, se na uzlu zvýší hodnota lokálních hodin o hodnotu d, kde d je dohodnutá konstanta, která je ve většině případů rovna jedné. Zpráva o dané operaci je následně označena právě touto novou hodnotou hodin C_i.
- R_2: pokud C_i jsou lokální hodiny uzlu, který přijal zprávu a C_m jsou hodiny přijaté zprávy, postupuje zpracování zprávy podle následujících pravidel:
  - 1. C_i=*max*(C_i,C_m)
  - 2. aplikujeme pravidlo R_1
  - 3. provede se operace obsažená ve zprávě

Skalární logické hodiny nejsou silně konzistentní, neboť logické lokální hodiny a globální logické hodiny nejsou striktně oddělené. Pro událost a_i a událost a_j a jejich skalární hodiny $C(a_i)$ a $C(a_j)$, kde $C(a_i) < C(a_j)$ nemusí nutně platit, že $a_i < a_j$. Může nastat situace, kdy N-tá událost uzlu A má menší hodnotu skalárních hodin než N-tá událost uzlu B, ale N-tá událost na uzlu A nenastala dříve než N-tá událost na uzlu B.

Vektorové hodiny

Absenci silné konzistence u skalárních hodin řeší hodiny vektorové. Stejně jako u skalárních hodin se nejedná o fyzický čas, ale o čítač operací. Na rozdíl od skalárních hodin není čítač realizován jako jedno kladné číslo pro jednu událost a proces, ale pro synchronizaci je využíván vektor, který obsahuje časové značky všech uzlů, tak jak je vidí každý konkrétní uzel samostatně. Tento vektor logických hodin se přikládá ke každé zprávě o události a pomocí něj dochází k synchronizaci logických hodin v celém systému. Vektor s časovými značkami všech uzlů pro danou operaci umožňuje porovnat pořadí všech operací pro vysílací i přijímací uzel a můžeme tvrdit, že vektorové hodiny jsou silně konzistentní. Obdobně jako pro skalární hodiny, i u vektorových hodin definujeme dvě pravidla, podle nichž se synchronizace řídí :

- R_1:Před provedením každé operace je provedena aktualizace lokálního vektoru, pro i-tou operaci to bude $vC_i[i] = vC_i[i] + d$, kde d je stejně jako v

případě skalárních hodin celočíselná hodnota typicky nastavená na hodnotu 1. Následně uzel pošle ostatním uzlům zprávu o operaci, kde zpráva je doplněna o celý vektor hodin.

- R_2:Po přijetí zprávy, která musí dle pravidla R_1 obsahovat vektor hodin vC_m, se provedou následující operace:
  - 1: aktualizace lokálních hodin pomocí vzorce: $vC_i[k] = max(vC_i[k],vC_m[k])$
  - 2: přidělení lokálního času pomocí pravidla R_1
  - 3: provedení operace z přijaté zprávy

Pomocí vektorových hodin může rozlišit několik vzájemných vztahů události :

- události jsou shodné, pokud pro všechna x platí: $vC_i[x] = vC_j[x]$
- událost vC_i nastala dříve nebo současně jako vC_j pokud pro všechna x platí: $vC_i[x] <= vC_j[x]$
- událost vC_i nastala dříve než událost vC_j pokud pro všechna x platí: $vC_i[x] < vC_j[x]$
- události vC_i a vC_j jsou konkurenční (souběžné) pokud platí všechna i,j neexistuje varianta kdy $vC_i < vC_j$ a zároveň neexistuje varianta $vC_j < vC_i$.

Pokud nastane poslední uvedený případ, tedy že jsou operace souběžné, není pomocí vektorových hodin možné určit pořadí procesů. Zde je třeba algoritmus rozšířit o prioritu uzlů. V případě shodné časové značky bude upřednostněn uzel s vyšší či nižší prioritou, podle scénáře, který bude dohodnut ve všech uzlech. Priorita může být například prosté *ID* uzlu, IP adresa či konstanta podle volby administrátora. Pro správné fungování priorit nesmí mít žádné dva uzly prioritu stejnou. Pro vztah mezi uspořádáním událostí a uspořádáním časových značek platí isomorfismus. Pokud událost a předchází událost b, pak pro časové značky těchto událostí platí $vC_a < vC_b$. Typickým využitím vektorových hodin je zajištění konzistence replikovaných dat.

Maticové hodiny

Maticové hodiny rozšiřují informaci z vektorových hodin na úplné uspořádání. K odesílané zprávě o události je připojena matice vektorů logických hodin. Jeden vektor je vektor logických hodin uzlu, který informaci odesílá, ostatní sloupce jsou doplněny o vektory logických hodin ostatních procesů ve stavu, v jakém byly uzlu odesílajícímu zprávu doručeny. Při odeslání zprávy tedy přijímací uzly vidí nejen stav počítadla odesílajícího uzlu, ale také jaké mají informace o počítadlech ostatních uzlů. Jelikož počet položek vektoru odpovídá počtu uzlů v systému a počet vektorů je také roven počtu uzlů v systému, matice bude vždy čtvercová n x n a budeme ji označovat $mT_i[]$. Tato matice vektorů logických hodin $mT_i[]$ reprezentuje pohled procesu P_i na globální stav logických hodin v systému. Jednotlivé prvky matice $mT_i[]$ mají následující význam:

- $mT_i[i,i]$ - obsahuje časové značky událostí které se udály lokálně
- $mT_i[i,j]$ - obsahuje lokální časové značky procesů $P_j(mt_j[j,j])$

- $mT_i[j,k]$ - obsahuje lokální časové značky reprezentující pohled na vztah lokálních hodin procesů $P_j(mt_j[j,j])$ a $P_k(mt_k[k,k])$

Jako pro každé logické hodiny, musíme pro maticové hodiny také definovat operace R_1 a R_2 :

- R_1: před provedením každé operace se provede navýšení stavu logických hodin v matici $mT_i[]$ následovně: $mT_i[i,i]=mT_i[i,i] + d$, kde $d > 0$, typicky se jedná o číslo 1.
- R_2: každá odeslaná zpráva bude doplněna o matici $mT_ms[]$. Při přijetí zprávy obsahující matici $mT_m[]$ uzlem P_j od uzlu P_i provedeme následující operace:
  - provedeme aktualizaci prvků $mT_i[i,k]$ pro všechna k v intervalu $[0..n]$ lokální matice prvky matice mT_m přijatými od uzlu P_j podle pravidla $mT_i[i,k] = max(mT_i[i,k],mT_m[k,j])$
  - provedeme aktualizaci všech dalších prvků lokální matice $mT_i[]$, kde pro všechny indexy k,l z intervalu $[0..n]$ podle pravidla: $mT_i[k,l] = max(mT_i[k,l],mT_m[k,l])$

Po provedení těchto pravidel je teprve možné přijatou zprávu zpracovat. Výhodou maticových hodin oproti vektorovým je fakt, že každý proces má kompletní přehled o stavu logických hodin všech procesů na všech uzlech a díky tomu i informaci o stavu zpracování jednotlivých operací.

5.6 Transakční zpracování operací

Jednou z důležitých předností distribuovaných systémů je jejich odolnost vůči výpadkům a možnost vyvažování zátěže v rámci systému. Tyto dvě vlastnosti jsou z velké části zajišťovány pomocí replikací, pro jejichž realizaci a zajištění zvoleného konzistenčního modelu je nutné transakční zpracování požadavků. Za použití transakcí je možné v rámci celého distribuovaného systému zajistit synchronní stav pro zvolený konzistenční model i při použití replikací.

5.6.1 Obecné vlastnosti transakcí

Transakci definujeme jako provedení jedné nebo více operací, které musí mít následující vlastnosti:

- *Atomicitu* - transakce je nedělitelná, buť je provedena celá a nebo není provedena žádná její část.
- *Konzistenci* - transakce nesmí porušovat pravidla zvolené konzistence nebo v případě databází integritních omezeních.
- *Izolovanost* - veškeré operace, které jsou prováděné uvnitř transakce jsou před okolním světem skryty až do provedení transakce, v případě chyby při vykonávání transakce je možné provést obnovu, která obnoví stav před započetím transakce a touto obnovou nejsou ovlivněny žádné jiné transakce nebo data.

- *Trvalost* - výsledek všech operací uvnitř transakce po jejím úspěšném dokončení a provedení je trvalý a tedy publikovaný vně transakce, kde s ním mohou dále pracovat další transakce.

V rámci transakce často požadujeme zachování pořadí prováděných operací. Toto pravidlo nemusí platit vždy a je možné tuto silnou podmínku u určitých operací, které je možno provádět paralelně, zlehčit či pominout, v závislosti na typu operace a návrhu systému. Vykonání transakcí se realizuje za pomoci žurnálu, tedy prostoru, kde jsou zaznamenány veškeré potřebné údaje o systému před započetím transakce a následně o jejím průběhu. Veškerá činnost jedné transakce se v rámci celého systému projeví teprve po jejím celém správném dokončení a potvrzení. V případě, že transakci není možné bezchybně vykonat, je celá transakce zrušena a systém obnoven ze žurnálu do stavu před započetím transakce. Vykonání transakce se může podle návrhu systému jednou či víckrát opakovat než se označí na daném uzlu za neproveditelné. Posledním nutným prvkem zpracování transakcí jsou zámky nad zpracovávanými daty. Při začátku vykonávání transakce jsou data, nad kterými transakce pracuje uzamčeny ostaním procesům. Důvodem je, stejně jako při ošetření kritické sekce, zajištění unikátního přístupu k datům tak, aby nedocházelo k jejich modifici v průběhu vykonávání operací transakce nad žurnálem. Zámek může být realizován jen pro čtení nebo pro čtení i zápis.

5.6.2 Transakce v distribuovaném prostředí

Kromě lokálního zpracování transakce pro jeden uzel se transakce používají i v distribuovaném prostředí, kde je situace složitější. Distribuovaná transakce je taková transakce, která se neprovede pouze lokálně na jediném uzlu, ale je nutné ji provést na více uzlech. Typickým systémem, který využívá distribuované transakce, je distribuovaný souborový systém, který disponuje replikacemi. Při požadavku změnit replikovaná data v distribuovaném souborovém systému je nutné provést jednu transakci s daty na všech uzlech, kde jsou replikovaná data uložena. Transakční zpracování je zde nutné ze dvou důvodů. Prvním důvodem je zajištění unikátního přístupu ke zpracovávaným datům a to nejen lokálně, ale napříč celým systémem, aby nedošlo ke změně zpracovávaných dat na jiném než lokálním uzlu. Druhým důvodem je zachování konzistence dat z pohledu uživatele, tedy aby změna byla provedena na všech uzlech nebo nikde. Z druhého uvedeného důvodu není možné prohlásit transakci za úspěšně dokončenou a provést trvalé potvrzení dříve, než úspěšnost provedení transakce potvrdí všechny zainteresované uzly. Potvrzování provedení transakcí je v distribuovaných systémech řešeno více algoritmy. Nejčastěji používané jsou dvoufázové potvrzování, třífázové potvrzování či dvoufázové potvrzování s hlasováním.

Dvoufázové potvrzovaní transakcí

Nejjednodušším způsobem realizace transakcí v distribuovaném prostředí je dvoufázové potvrzování. Před započetím zpracovávání distribuované transakce je mezi uzly, které se dané transakce budou účastnit, zvolen koordinátor. Typicky se jedná o uzel, který transakci inicializuje. Koordinátor rozešle ostatním uzlům, které se mají transakce zůčastnit zprávu s žádostí o potvrzení připravenosti realizovat

transakci. Pokud dostane všechny kladné odpovědi, transakce se začne vykonávat. Po úspěšném dokončení provedení lokální transakce se koordinátor dotáže všech ostatních uzlů na výsledek zpracování transakce na jednotlivých uzlech. Tím získá potřebnou informaci, zda je možné transakci dokončit a výsledek trvale zveřejnit. Koordinátor po úspěšném dokončení své lokální transakce rozešle dotaz ostatním uzlů, zda se dokončení a tedy i zveřejnění výsledku transakce provedlo v pořádku. V případě všech kladných odpovědí je transakce prohlášena za dokončenou. V případě, že během zpracování výše uvedené posloupnosti operací dojde na jakémkoli uzlu k chybě, je celá transakce na všech uzlech zrušena. Podstatnou nevýhodou dvoufázového potvrzování transakcí je skutečnost, že tento způsob zpracování transakcí je blokující od počátku zahájení transakce koordinátorem až do doby, než koordinátor obdrží zprávu s potvrzením či chybou od všech zúčastněných uzlů. Tento nedostatek řeší třífázové potvrzování transakcí.

Třífázové potvrzování transakcí

Třífázové potvrzování transakcí výchází z dvoufázového potvrzování, ale nově zavádí časové limity na jednotlivé operace a zároveň operaci předběžného zápisu. V prvním kroku koordinátor rozešle dotaz na připravenost uzlů provést transakci. Pokud kterýkoli z uzlů odpoví záporně nebo neodpoví v časovém limitu je transakce zrušena. Ve druhém kroku koordinátor na základě všech kladných odpovědí z kroku jedna rozešle zprávu s žádostí o provedení předzápisu. Pokud tuto zprávu neobdrží kterýkoliv uzel, který má operaci vykonat, považuje ji v rámci dohodnutého času za doručenou a provede předzápis, přestože zprávu neobdržel a na jejím závěru posílá koordinátorovi potvrzení o provedení. Pokud koordinátor byť jen od jediného uzlu neobdrží do stanoveného času odpověd, celou transakci zruší. Pokud koordinátor obdrží včas všechny kladné odpovědi, přistoupí k třetí fázi. Ve třetí fázi koordinátor odešle příkaz k provedení trvalého zápisu. Stejně jako v předchozím případě pokud vykonávající uzel neobdrží zprávu s příkazem na trvalý zápis v rámci stanoveného času, považuje zprávu za doručenou, provede zápis a odesílá potvrzení o provedení transakce. Pokud koordinátor v dohodnutém čase přijme od všech uzlů kladná potvrzení, prohlásí transakci za úspěšně dokončenou. Pokud obdrží byť jen jednu zápornou odpověď nebo nějaká odpověď nedorazí ve stanovený čas vůbec, je celá transakce zrušena a je provedena obnova. Třífázový způsob potvrzování sice řeší nedostatky dvoufázového potvrzovacího protokolu, ale kvůli své komunikační složitosti se téměř nepoužívá.

5.7 Transparentnost

Transparentnost je vlastnost distribuovaného souborového systému, která eliminuje vazbu na fyzickou topologii systému a zároveň skýtá existenci násobných prvků v distribuovaném systému jakou jsou replikovaná data či uzly nebo násobné cesty v systému . Uživatel neví, na kterém uzlu jsou jeho data uložena a ani skutečnost, zda jsou jeho data replikována. V rámci transparentního přístupu k datům se zavádí speciální jmenný prostor, který svoji strukturou nemusí odpovídat fyzickému rozložení dat, ale mapuje se na něj. Jednotlivá data v distribuovaném souborovém systému jsou sdružována do skupin, které se běžně označují jako svazky. Tyto svazky jsou nejmenší replikovatelnou jednotkou a mapují se z

pohledu uživatele na adresář. Transparentnost přístupu bez nutnosti znát reálný název či uložení dat nám zároveň umožňuje jednotlivé svazky libovolně přesouvat mezi jednotlivými uzly distribuovaného souborového systému a to dokonce za chodu. I tento přesun je pro uživatele transparentní. Transparentnost dat je typicky řešena pomocí překryvných metadat, které existují v podobě databáze, která mapuje fyzická data na jednotlivých serverech na soubory a adresáře viditelné uživatelem. Toto mapování nám navíc umožňuje data dělit na menší datové bloky, se kterými je možné na pozadí systému pracovat samostatně. Bloky jednoho souboru uživatele mohou být rozděleny na více serverů, což zvyšuje jejich dostupnost. Bloky mohou být čteny paralelně což zvyšuje propustnost systému a rychlost odezvy uživatele a v neposlední řadě mohou být použity i k úspoře zabraného datového prostoru v rámci deduplikace datových bloků, jak bude popsáno v podkapitole 7.7.3. Dalším přínosem transparentnosti v distribuovaných systémech je zvyšování míry zabezpečení, neboť není zcela jednoduché pro uživatele či útočníka dohledat konkrétní server či datový zdroj, který by mohl sloužit jako cíl útoku.

5.8 Bezpečnost

Důležitou součástí všech, tedy nejen distribuovaných, systémů, je bezpečnost [37]. V případě distribuovaných systémů je však tato otázka o to ožehavější, neboť většina distribuovaných systémů, především pak geograficky rozsáhlých, ke své komunikaci nevyužívá jen uzavřené lokální sítě, ale i veřejný internet. V rámci internetu je důležité zajistit bezpečnost jak samotných serverů, tak bezpečnost, konzistenci a důvěryhodnost dat přenášených po veřejných částech počítačové sítě a v neposlední řadě i bezpečnost uživatelů a jejich přístupových údajů. Systémů a způsobů zabezpečení je mnoho, ale vždy s sebou kromě zvýšení úrovně zabezpečení přinášejí i vyšší režii a to jak z hlediska datových přenosů tak i vyšší nároky na potřebný výpočetní čas a snížení jednoduchosti použití pro uživatele [38], [39].

5.8.1 Šifrování

Nejjednodušší, nejstarší, a také nejméně bezpečné systémy, vyžadovaly jen znalost jména a hesla, kde heslo bylo odesíláno po síti v nezašifrované podobě. Potenciálnímu útočníkovi stačilo zachytit komunikaci mezi klientem a serverem a s pomocí zachyceného hesla mohl do systému proniknout. Na základě tohoto podstatného nedostatku se začalo při přenosu citlivých dat počítačovou sítí používat šifrování.

Symetrické šifrování

Prvním způsobem je šifrování symetrickým klíčem. Při tomto způsobu se k šifrování i dešifrování informace používá stejný klíč. Tento klíč musí být vždy utajen. Toto řešení je jednoduché na implementaci a velice rychlé při použití. Nevýhodou symetrického šifrování je nutnost mít před započetím přenosu sdílený šifrovací klíč na obou komunikujících uzlech. Příkladem symetrických šifer jsou AES[41], Blowfish[42] nebo DES[43].

Asymetrické šifrování

Druhým způsobem je šifrování asymetrické. Při tomto způsobu šifrování existují dva klíče. Jeden neveřejný klíč, který musí zůstat utajen. Od neveřejného klíče je odvozen veřejný klíč, který může být komukoliv dostupný. Odesílaná data jsou šifrována veřejným klíčem, ale dešifrována neveřejným. Veřejný klíč tedy může být volně dostupný, neboť na základě jeho znalosti není možné informaci dešifrovat. Tento způsob šifrování je bezpečnější, ale výpočetně i časově náročnější. Příkladem asymetrických šifrování jsou RSA[44] nebo Diffie-Hellman[45].

Kombinované šifrování

V současné době se často využívají kombinace obou způsobů šifrování. Pomocí asymetrického šifrování s veřejným klíčem se vymění mezi stranami symetrický šifrovací klíč a pomocí něj je šifrována veškerá následná komunikace. Díky tomu je komunikace jak bezpečná tak i rychlá.

5.8.2 Autentizace

Šifrování je jen jednou složkou zabezpečení. S ohledem na skutečnost, že distribuované souborové systémy obsahují data uživatelů nebo celých systémů, je nutné přístup k nim omezovat a řídit. Druhou důležitou složkou bezpečnosti distribuovaných systémů je bezpečný způsob autentizace a autorizace. Autentizace je proces, jehož cílem je prokázání identity uživatele, systému nebo služby.

Jméno a heslo

Dalším jednoduchým způsobem autentizace je ověření jménem a heslem, jak bylo zmíněno v úvodu této kapitoly. Tento způsob autentizace se stále hojně využívá, byť se nejedná o zcela bezpečný způsob autentizace a to hned z několika důvodů. Prvním již zmíněným problémem je možnost odposlechnutí hesla. Tento nedostatek je možné řešit pomocí šifrování komunikace. Dalším a v poslední době velice často zneužívaným problémem je možnost uhodnutí hesla. V takovém případě útočník neodposlouchává komunikaci, ale sám se aktivně snaží do systému přihlásit za pomoci hesla, které může být generované, vybrané z existujících slovníků a nebo se velmi často jedná o kombinaci obou způsobů. Ochranou je správná volba hesla a omezení počtu neúspěšných přihlášení.

Kerberos

I při použití vhodného šifrování stále hrozí možnost úniku hesla a to prolomením šifrované komunikace. Ideální by bylo takové ověření, při jehož použití nebude heslo nikdy přenášeno ani v zašifrované podobě prostřednictvím počítačové sítě. Tento mechanismus je použit v systému Kerberos[36]. Systém Kerberos přijme lokálně na stanici heslo od uživatele a s použitím dalších údajů jako je čas, uživatelské jmeno či doména provede nad těmito údaji sérii matematických operací jejichž výstup je teprve přenášen na server k ověření. Server má ve své databázi k dispozici otisk hesla uživatele opět v podobě výsledku matematické funkce a provede ověření z přijatých a uložených údajů další matematickou funkcí jejichž výsledkem je ověření či neověření uživatele, ale bez nutnosti heslo v jakékoliv

podobě přenášet počítačovou sítí. Další výhodou systému Keberos je fakt, že na základě prokázání totožnosti je klientovi vygenerován časově omezený tiket, pomocí kterého se může ověřovat vůči dalším službám, jak bude popsáno v kapitole 7.3.

Certifikáty a X.509

Další možnost autentizace vychází ze standardu X.509[46], který definuje principy fungování systémů založených na existenci veřejného klíče. Nejčastějším příkladem použití jsou certifikáty. V systému certifikátů jsou definované certifikační autority, označované jako CA. Certifikační autorita je organizace, která vlastní certifikát, kterým jsou podepsané vydávané certifikáty generované na základě privátního klíče, který dodá žadatel o certifikát. Tím je možné ověřit, zda daný certifikát byl podepsán uváděnou certifikační autoritou. Pokud tomu tak je a dané certifikační autoritě důvěřujeme, důvěřujeme i informaci, že daný uživatel nebo systém, který provedl autentizaci, prokázal svoji totožnost, neboť je ověřitelný danou certifikační autoritou. Výhodou systému certifikátu je i to, že tvoří hierarchický model, tedy je mezi certifikáty možné definovat vazby a hierarchie, například podle struktury organizace. Systém certifikátů nemusí sloužit jen k autentizaci uživatele či služby, ale může sloužit i například k šifrování komunikace. Šifrování komunikace pomocí certifikátů se nejčastěji používá pro zabezpeční komunikace s WWW serverem pomocí protokolu HTTPS, což je zabezpečené rozšíření komunikačního protokolu HTTP. V případě HTTPS komunikace mají certifikáty dvojí úlohu, jednak umožňují šifrovat komunikaci, druhak umožňují identifikovat server, ke kterému se připojujeme a to právě na základě ověření platnosti certifikátu u certifikační autority.

V distribuovaném systému můžou certifikáty zastávat všechny tři své možné funkce, tedy šifrovat komunikaci, klientovi umožnit ověřit, že protistrana, ke která se připojuje je opravdu tím za koho se vydává a samozřejmě prokázat totožnost vlastníka certifikátu.

Další způsoby

Existuje celá řada dalších systémů autentizace jako jsou OTP, tedy jednorázová hesla[47], systémy ověření na základě biometrických údajů, jako je otisk prstu, sken sítnice či analýza hlasu. Většina uvedených způsobů ověření je ale zaměřená na identifikaci člověka, zatímco v distribuovaných systémech se prokazují jak uživatelé ve formě reálného uživatele - člověka, tak i služby či servery, pro které tyto autentizační metody nejsou vhodné.

5.8.3 Autorizace

Po úspěšně provedené autentizaci uživatele je nutné uživatele autorizovat. Proces autorizace má za cíl subjektu, který už prokázal svoji totožnost, umožnit pracovat se systémem dle nastavených oprávnění. U distribuovaných systémů se rozhodujeme zda daný subjekt má či nemá právo danou službu používat. Pro řešení stanovení oprávnění existují dva možné přístupy a to *volitelné řízení přístupu* označované jako DAC z anglického "Discretionary Access Control" a *povinné řízení přístupu* označované jako MAC z anglického "Mandatory Access Control".

Volitelné řízení přístupu - DAC

Základní myšlenka volitelného přístupu řízení je v tom, že vlastník dat může s těmito oprávněními manipulovat, například udělovat oprávnění dalším subjektům. Tento přístup řízení má zásadní nedostatek v tom, že administrátor celého systému není schopen oprávnění řídit centrálně. Pro volitelný způsob řízení přístupu jsou typické dva submodely, prvním je přístup na principu unixových práv, druhým je rozšířený model ACL z anglického Access Control List.

Systém založený na principu unixových práv je implementačně jednodušší neboť pro data definuje tři možné typy vlastnictví což jsou :

- Uživatel - jeden konkrétní uživatel
- Skupina - jedna konkrétní skupina uživatelů
- Ostatní - všichni ostatní, kteří nejsou obsažení ani v položce uživatel ani nejsou členy skupiny.

K této trojici oprávnění jsou definována tři základní práva nad daty:

- r - právo číst soubor
- w - právo modifikovat soubor
- x - právo spouštět soubor

Tato trojice může být podle implementace systému rozšířena o další speciální práva.

Unixový model práv má svá omezení, která jsou více zjevná v systémech s větší členitostí oprávnění. Například v situaci, kdy chceme pro několik uživatelů nebo skupin nastavit různé úrovně oprávnění, se základními unixovými právy nevystačíme. Proto vznikl systém rozšířených práv, který je založen na seznamech přístupu a označovaný jako ACL. ACL rozšiřuje základní model unixových práv a umožňuje k jednomu souboru přiřadit více uživatelů a skupin a pro každého uživatele či skupinu definovat vlastní úroveň oprávnění. Tento model nastavování oprávnění je pro distribuované souborové systémy typický, neboť se zpravidla jedná o rozsáhlé systémy s větším množstvím uživatelů a tedy i s více možnými úrovněmi oprávnění.

Povinné řízení přístupu - MAC

Nedostatečnou kontrolu nad nastavováním oprávnění při použití modelu volitelného řízení přístupu řeší model povinného řízení. Při použití modelu povinného řízení je veškeré nastavení oprávnění řízené centrálně administrátorem. Pro řízení přístupu jsou stanoveny dvě entity a definovány vztahy mezi nimi. První entitou je subjekt, což je uživatel, služba nebo proces. Druhou entitou je objekt, což je soubor nebo obecně zdroj. Administrátor definuje schopnost subjektů pracovat s objekty a vztahy subjektů mezi sebou. Tyto vztahy definují oprávnění a subjekt ani objekt není schopen je měnit. Dodržování zvoleného nastavení je kontrolováno na úrovni jádra operačního systému, čímž je zajištěno jejich striktní dodržování. Jednotlivé objekty i subjekty od sebe mohou být úplně odděleny. Výhodou tohoto přístupu je, že pokud nějaká služba bude kompromitována, útočník nebude moci

do celého systému, ale jen ke zdrojům, které administrátor přidělil napadené službě a toto nastavení nebude moci nijak změnit ani obejít. Příkladem implementace tohoto přístupu řízení oprávnění je projekt SELinux[48].

5.9 Současné distribuované souborové systémy

V současné době existují distribuované souborové systémy v podobě OpenSource, ale i jako komerční řešení [49]. V následující části textu budou nejrozšířenější a nejpoužívanější distribuované souborové systémy v krátkosti představeny. Při popisu vlastností bude kladen důraz na ty vlastnosti, které budou stěžejní pro další části této práce, tedy rozšiřitelnost, implementace pro operační systémy existující na mobilních zařízeních a podporující RW replikaci.

5.9.1 Network File System - NFS

Nejstarším představitelem distribuovaných souborových systémů je NFS[50]. NFS je distribuovaný souborový systém pracující v architektuře klient - server. V současné době už došel ve svém vývoji do čtvrté verze. Konfigurace serveru se provádí textovým souborem, typicky na unixových systémech /etc/exports. V tomto souboru je uvedeno, která data budou sdílena, z jakých IP adres nebo skupin IP adres bude možné k serveru přistupovat, jak bude prováděno přemapovávání uživatelů, zda se bude jednat o synchronní či asynchronní přístup, zda bude možné data zapisovat či jen číst atd. Data, která jsou sdílena, jsou sdílena ve formě úplných cest k adresářům, které tyto data obsahují a tyto cesty jsou v nezměnitelné podobě uvedeny i na klientovi. NFS se mezi verzemi 2,3 a verzí 4 podstatně změnil a z původního síťového souborového systému vznikl skutečně distribuovaný souborový systém.

NFS verze 4 je následník systému NFS verze 2 a 3. NFS verze 2 a 3 nejsou distribuovaným souborovým systémem, ale jen síťovým souborovým systémem. V těchto verzích systém nepodporoval žádnou formu ověření, kromě ověření shody UID uživatele a filtrace přístupu na základě IP adres, navíc data nemohla být replikována, ale mohla být sdílena pouze z jediného zdroje. V implementaci NFS verze 4 se autoři snažili odstranit nedostatky předchozích verzí. Zásadní změnou je začlenění podsystémů mount a lock do hlavní části programu. NFS verze 4 stále využívá volání RPC, ale již komunikuje na jediném portu TCP 2039, což umožňuje jeho použití v kombinaci s firewallem. Pro komunikaci je možné použít TCP i UDP protokol. Pro NFS verze 4, které je nasazované i mimo lokální sítě je výhodnější používat protokol TCP.

K ověření je použit systém Kerberos, což je výrazný posun k lepšímu proti předchozím verzím, kde jediná forma autentizace byla na základě shody UID. Pro autorizaci je v NFS verze 4 možné použít jak unixová práva, tak jejich rozšíření ve formě ACL. Další podstatnou změnou je zavedení podpory kódování UTF-8, což umožňuje spolupracovat s novějšími verzemi Microsoft Windows. Ve specifikaci protokolu je také uvedena možnost replikace a zrcadlení dat. Dalším důležitým rozšířením je možnost cachovat datové soubory. Povolení cachovat data vydává klientovi server na určitou, předem stanovenou dobu. Zároveň jej může v případě modifikace dat jiným uživatelem odebrat. V NFS verzi 4 je již také nově implementován protokol pro uzamykání souborů, který lépe řeší problematiku

konkurenčního zápisu dat. Hlavní změnou je, že zamykání předpokládá chybovost sítě a tak nedochází k trvalému zamykání souborů jako u verzí 2 a 3, ale zámek má pouze časově omezenou platnost.

5.9.2 Self-certifying - SFS

SFS[51] je distribuovaný souborový systém vycházející z NFS. SFS tvoří bezpečnostní nadstavbu NFS a řeší některé jeho nedostatky jako je především autentizace a šifrování komunikace. Ke své činnosti potřebuje tento systém NFS verze 3 na straně klienta i serveru. Použití NFS je však pouze lokální na lokálním rozhraní. Veškerou komunikaci přes síť pak obstarává SFS démon, který funguje pouze v uživatelském módu a předává požadavky na modulu NFS jádra operačního systému. Komunikace prostředím sítě je zabezpečená pomocí šifrování, díky čemuž je SFS možné bezpečně provozovat i na veřejných sítích. Komunikace probíhá pomocí TCP protokolu na definovatelném portu, což umožňuje snadné využití firewallu a zároveň spolehlivou komunikaci i na nestabilních linkách díky potvrzování komunikace na úrovni TCP protokolu. Oproti NFS je na klientovi sdílený svazek vždy napojen do jednoho z podadresářů adresáře /sfs. Podadresář, do kterého je svazek připojen, je určen pomocí *@Location%port,HostID*. Obsah /sfs adresáře se pro jednotlivé uživatele liší, podle toho, k jakým serverům jsou autorizovaní. SFS nevyužívá k autentizaci uživatelů prostředků třetích stran. Autentizace je prováděna pomocí klíčů generovaných administrátorem a instalovaných u konkrétních uživatelů.

5.9.3 Microsoft Distributed File System - DFS

DFS[52] je distribuovaný souborový systém vyvíjený společností Microsoft. DFS není samostatná implementace souborového systému, ale jedná se o nadstavbu nad protokoly Server Message Block či jeho novější varinatu Common Internet File System. DFS funguje jako agregátor síťových sdílení. DFS je úzce svázán s Microsoft Active Directory a využívá jeho služeb, jako jsou především autentizace a autorizace. V rámci autorizace jsou dostupná rozšířená práva ACL. Kořen DFS je dostupný jako samostatné sdílení v rámci Active Directory. Data, která DFS poskytuje, jsou poskytována jednotlivými lokálními souborovými servery. Data nejsou replikovaná.

Server Message Block - SMB

SMB[53] byl vytvořen jako síťový souborový systém pro operační systém Microsoft DOS a Microsoft Windows. Postupem času byl přeportován na další operační systémy a v současné době se s ním setkáme téměř na všech operačních systémech a to v podobě OpenSource implementace serveru s názvem Samba[55] a SMB klienta, jako modul jádra operačního systému Linux. Vzhledem k tomu, že byl tento systém původně určen pouze pro operační systémy Microsoft DOS a Microsoft Windows, byly jeho původní vlastnosti přizpůsobeny těmto operačním systémům například délkou a strukturou jména souborů. Postupným vývojem se systém pojmenovávání souborů rozšířil i na těchto operačních systémech, takže nyní se jedná o plně použitelné řešení.

V operačním systému Microsoft Windows je SMB implementován jako neoddělitelná součást operačního systému. Naproti tomu na ostatních systémech je SMB implementován jako Samba server, který funguje v uživatelském módu. Stejně jako implementace serveru, je rozdílná i implementace a vlastnosti klienta. Na operačních systémech Microsoft Windows je možné sdílenou složku připojit pouze jako další disk. V ostatních operačních systémech je možné SMB připojit jako libovolný adresář. Proti NFS je v SMB zvýšené zabezpečení i možnosti nastavení. Jednotlivé sdílené složky mohou být sdíleny pod jiným jménem než se jmenují ve skutečnosti na serveru a jako identifikátor připojení slouží pouze toto virtuální jméno a identifikace stroje, který požadovaná data sdílí. Autentizace klienta je prováděna pomocí uživatelského jména a hesla, které může být poskytováno lokální autentizační databází serveru nebo centrálně prostřednictvím Active Directory. Systém SMB nepracuje s UID vlastníků souborů, ale s jejich přihlašovacím jménem, takže uživatel může mít na více systémech různé UID, což odstraňuje další nedostatek systému NFS. SMB v implementaci Samba obsahuje překlad uživatelských jmen – takzvané aliasy. Ty slouží jednak pro případ, že uživatel má ve více systémech různá uživatelská jména a druhak řeší historický problém pojmenovávání uživatelů. Prvotní verze operačního systému Microsoft DOS a Microsoft Windows posílaly při přihlašovaní uživatelská jména ve tvarech zadaných uživatelem, ale bez rozlišení malých a velkých písmen. Proto bylo nutné vytvářet překladové tabulky pro konverzi na systémy, kde jsou velikosti písmem rozlišovány, jako je například Linux.

Stejně jako NFS verze 2 a 3 i SMB postrádá možnost on-line replikace dat. SMB také postrádá možnost cachovat jednotlivé soubory. Dalším nedostatkem SMB, především s ohledem na použití na mobilních zařízeních, či obecněji na zařízeních s nestabilním datovým připojením je skutečnost, že systém si určitou dobu pamatuje stav síťových zařízení a nereaguje na změny konfigurace do úplného restartu zařízení. Tento nedostatek se projevuje především na operačních systémech Microsoft Windows. Kladnou stránkou SMB je, že komunikuje pouze na portech 137, 139, 445 a protokoly TCP a UDP, což umožňuje snadné nasazení firewallu a je možné zvýšit tak bezpečnost systému omezením přístupu už na síťové vrstvě.

Common Internet File System - CiFS

CiFS[54] je následovníkem protokolu SMB, se kterým je zpětně kompatibilní. Někdy se také uvádí, že se jedná o dialekt SMB protokolu. V prostředí unixových systémů se jedná o dvě implemetace stejného protokolu, kde CiFS je novější a řeší nedostatky starších implementací SMB protokolu. Nejzásadnější změnou je zavedení podpory šifrování požadavků přenášených datovou sítí. V současné době se na unixových systémech používá už výhradně CiFS implementace.

5.9.4 Andrew File System - AFS

AFS[59] je jeden z nejkomplexnějších existujích distribuovaných souborových systém s velkým množstvím možností a nastavení. Systém AFS vznikl z projektu Andrew[58] na Carnegie Mellon University v roce 1982. V současné době existuje AFS verze 3 a to i ve více implementacích. Asi nejrozšířenější je verze vytvořená v rámci projektu OpenAFS[60]. AFS se snaží návrhem i implementačně pokrýt

většinu vlastností, které jsou pro distribuovaný souborový systém stěžejní, především bezpečnost, stabilitu, škálovatelnost, transparentnost a možnost replikace dat. Celý projekt AFS a jeho klony jsou díky rozsáhlosti implementace značně svázané původním návrhem a jen velmi pomalu se do nových verzí vkládají nové vlastnosti.

AFS je založené na architektuře server-klient. Klienti jsou vzájemně rovnocenní. Serverů je několik typů podle činnosti, kterou v rámci AFS systému vykonávají. Nejpoužívanější servery jsou uvedeny v tabulce 5.2 s vysvětlením jejich úlohy v distribuovaném souborovém systému.

Název	**Popis**
bosserver	Stará se o konfiguraci systému a spouští další funkční servery
ptserver	Obsluhuje metadata a interní databázi uživelů a skupin
vlserver	Poskytuje informace o svazcích a jejich replikách
fileserver	Obsluhuje požadavky na operaci s daty, tedy soubory a adresáři
volserver	Řeší mapování svazků do virtuální adresářové struktury
buserver	Realizuje zálohování
kaserver	Interní autentizační server

Tabulka 5.2: Nejčastěji používané typy serverů v systému AFS

Všechny servery stejného typu, provozované na různých strojích, jsou vzájemně rovnocenné. Server je kompletně implementován v uživatelském módu operačního systému. Klient kombinuje části v uživatelském módu s modulem jádra.

Seznam uživatelů AFS je uložen v interní proprietární databázi, která je online synchronizováná na všech serverech v AFS skupině. Pro autentizaci uživatelů je možné využít interní kaserver, ale častěji je autentizace realizována pomocí externího řešení v podobě systému Kerberos. Po ověření přiřadí Kerberos klientovi tiket, který je pro potřeby AFS transformován na token, sloužící k další autentizaci. Tuto vlastnost AFS od Kerbera převzalo. Její výhodou je, že není po klientovi opakovaně požadováno zadání hesla a ani není heslo po dobu připojení klienta uloženo v paměti a přesto je zachována možnost opakovaného použití autentizačních údajů, například na přístup na další AFS servery. Kromě systému ověření pomocí vstupních ticketů/tokenů, převzalo AFS od Kerbera i členění do skupin, které jsou označované jako *Cell* nebo také domény. Tyto domény nemusejí korespondovat s DNS doménami, ale definují uzavřenou skupinu serverů, které společně tvoří jeden AFS systém a vzájemně si důvěřují. To umožňuje systém AFS používat v rozsáhlých sítích a tím lépe chránit data proti poškození a zvyšuje dostupnost systému. Domény v AFS mohou tvořit hierarchii a je možné nastavit vztah jednosměrné či obousměrné důvěry mezi jednotlivými doménami.

V AFS definuje vlastní systém oprávnění, který je klasickému zabezpečení na klientské stanici nadřazený. Tento systém podporuje ACL systém, takže dovoluje i nastavení složitější struktur práv. K nastavení práv se používají speciální prostředky dodávané s AFS. Klasické příkazy operačního systému pro nastavení práv a vlastníků souborů či adresářů lze samozřejmě použít, ale ty jsou dostupné jen z důvodu dodržení kompatibility s klasickými souborovými systémy a jejich implementace pomocí VFS. ACL práva AFS jsou právům klasického souborového systému vždy nadřazena.

Replikace dat je v AFS řešena ve formě Master RW repliky. V systému je možné mít replikovaná data na více serverech. Replikační jednotkou u AFS je svazek. Jeden svazek může být replikovaný na více serverech, ale vždy pouze jediná replika svazku je zapisovatelná. Pokud klient chce číst data může použít libovolnou repliku, ale pro zápis musí použít tu, která je zvolena administrátorem a neexistuje automatická možnost její změny v případě výpadku. Replikace je řešena jako vynucená akce administrátorem. Data nejsou na jednotlivé repliky distribuována v čase změny či se zpožděním automaticky, ale až po ručním zásahu, což možnost aktivního používání replik jako nástroje na optimalizaci výkonu a zvýšení rychlosti odezvy systému pomocí volby nejvhodnější repliky, vylučuje. Přínosem replikace v AFS je zvýšení odolnosti dat proti výpadkům. Zvýšení dostupnosti dat z hlediska rychlosti je pomocí Master RW replikace možné docílit pouze na specifických vzorcích dat, která se často nemění. Prioritu existujících replik je možné pro jednotlivé klienty ovlivňovat, ale pouze ručním zásahem administrátora, neděje se tak automaticky s ohledem na aktuální stav a vytížení datové sítě.

Stejně jako Kerberos, je i AFS velice citlivé na nepřesnosti v nastavení času a na ztrátovost linky, což se projevuje výrazným zpomalením chodu celého systému při čekání na odpověď ze serveru či klienta. AFS vyžaduje obousměrnou komunikaci, neboť využívá takzvaných callback volání. Tento typ komunikace není iniciován klientem směrem k serveru, ani serverem proti serveru, ale jedná se o situaci, kdy server kontaktuje klienta. Typickým příkladem je ověření, zda je klient ještě stále aktivní při delší nečinnosti. Tím, že toto volání je realizované pomocí nově otevřeného spojení, je problematické plnohodnotně provozovat AFS systém v sítích, které jsou od serverů odděleny pomocí NAT technologie[61]. AFS je implementováno jako množina několika procesů na jednom serveru. Každý tento server naslouchá na své definované adrese a portu. Pro svoji komunikaci zatím AFS používá pouze protokol UDP. To sebou přináší mnohá výkonová omezení, neboť UDP, jakožto nepotvrzovaný protokol, není pro systém přenosu souborů vhodný a správnost a úplnost doručení všech částí přenášených dat je nutné implementovat v aplikaci, což sebou nese nemalé zpoždění, především na méně stabilních linkách jakou jsou mobilní připojení. Implementace komunikace pomocí TCP protokolu je již dlouhou dobu plánována, ale jedná se o tak zásadní zásah do klienta i serveru, že zatím nebyl realizován. Přesto, že AFS využívá ke své komunikaci více portů, je možné jej provozovat na serveru s podporou firewallu neboť seznam portů je konečný a předem daný.

Klient systému AFS umožňuje používat cache jak metadat souborů tak souborů samotných. Použití cache snižuje nutný přenos dat a zvyšuje rychlost odezvy pro klienta. Tato cache je implementována nad klasickým souborovým systémem jako samostatně připojený disk. Není však možné, jako podkladový souborový systém, použít všechny souborové systémy, jako například XFS[15]. Důvodem je nekompletní či rozdílná implementace některých VFS.

AFS neumožňuje libovolné připojení ani exportovaných svazků na serveru, ani libovolný přípojný bod na klientovi, jako tomu je u NFS. Na serveru jsou exportovaná data vždy uložena v adresářích */vice* a není možné je měnit. U klienta je celý AFS systém mapován do adresáře */afs*. AFS je stabilní a dobře zabezpečený systém, který disponuje základními vlastnostmi, které od distribuovaného souborového systému očekáváme, tedy replikací dat a tím zvýšenou bezpečností

a dostupností zdrojů, zabezpečení dat, škálovatelností do šířky – neboť je možné servery přidávat i ubírat za chodu systému a přenositelností, neboť server byl portován na řadu operačních systémů. Negativní vlastností je neúplná kompatibilita jednotlivých implementací pro různé architektury. Dalším negativem je návrh architektury a implementace, která neumožňuje snadnou modifikaci systému a jeho optimalizaci pro použití na mobilních zařízeních. Výrazným nedostatekem je také omezení maximální velikosti jednoho svazku. AFS nepodporuje svazky větší než 2TB, což je při velikosti dnešní datových skladů, které i u menších organizací dosahují velikosti desítek TB, omezující. Dva nejzásadnější nedostatky, především z pohledu nasazení v geograficky rozsáhlých systémech a na mobilních zařízeních jsou stávající systém replikací s jednou zapisovatelnou replikou, navíc aktualizovanou jen na základě pokynu administrátora, a neuspokojivá rychlost. Neboť i na síti o rychlosti 1Gbps a při použití diskových polí s dostupnou rychlostí čtení až 200MB/s dosahuje AFS, při zachování konzistence a bezpečnosti přenášených dat, rychlosti pouhých 7MB/s, jak bude prokázáno na testech v kapitole 7.9. I přes tyto nedostatky bude pro tuto práci OpenAFS použit jako referenční systém pro porovnání s nově navrženým systémem KIVFS, neboť se aktuálně jedná o nejčastěji používaný systém, který disponuje nejvíce implementovanými vlastnostmi, které jsou od distribuovaného souborového systému požadovány.

5.9.5 Constant Data Availability - Coda

Coda[56] je distribuovaný souborový systém, který vznikl v 80-90 letech 20. století a svým návrhem vychází ze systému AFS. Stejně jako u AFS se jedná o mnohem komplexnější a propracovanější systém než jsou ostatní systémy uvedené v předchozích kapitolách. Hlavní motivací návrhu a implementace systému Coda byl redesign a doplnění funkcí existujícího systému AFS. Coda byla přímo navržena jako distribuovaný souborový systém a nemá za sebou přechod ze síťového souborového systému a nejedná se o pouhou nadstavbu, ale samostatný distribuovaný souborový systém. Hlavním přínosem je zavedení možnosti replikace dat na více serverů. Klienti systému Coda jsou rovnocenní a připojují se k serverům, které jim poskytují data. Klient v systému Coda disponuje podporou lokální cache. To umožňuje jednak šetřit přenosy, neboť dvakrát požadovaný soubor, který nebyl změněn, je přenášen ze serveru jednou. Druhak je cache použita jako nástroj na realizaci podpory off-line operací, které umožňují klientovi pracovat s částí dat i v případě, že žádný server není právě dostupný. Veškeré provedené změny, které vznikly během fungování v off-line režimu jsou ukládány do lokální cache v tar formátu[62] a po obnovení spojení a přechodu systému do on-line režimu jsou tato data přesunuta na server, kde se provede kontrola provedených operací z pohledu dodržení konzistenčního modelu a operace neodporující konzistenčnímu modelu jsou provedeny. Může nastat situace, kdy by provedení požadovaných operací porušovalo konzistenční model. Typickým příkladem je situace, kdy data nad nimiž má být provedena některá z off-line operací byla na serveru v době nedostupnosti klienta změněna jiným uživatelem. V takovém případě není možné jednoznačně rozhodnout o provedení operace a systém rozhodnutí přenechá na uživateli, který je vyzván k rozhodnutí jak se má v konkrétních případech postupovat.

Systém Coda umožňuje použití více serverů, která mohou poskytovat unikátní nebo replikovaná data. Servery mohou mít jednu ze dvou rolí, a to Master a

Slave, což definuje i rozdílný způsob jejich chování. Servery kromě dat obsahují i metadata, která jsou uložena odděleně v databázi RVM[57]. Tato databáze tvoří virtuální prostor v paměti, kde jsou metadata systému uchovávána a tento obsah je zrcadlen na disk, aby mohl být při startu systému načten do paměti. Coda načítá při startu do paměti vždy celý obsah RVM a to výrazným způsobem prodlužuje start jednotlivých uzlů. Seznam uživatelů je uložen v interní systémové databázi. Tato databáze je typu single-master a na rozdíl od OpenAFS, které bylo popsané v předchozí kapitole 5.9.4, Coda nebrání lokálním změnám databáze na jednotlivých uzlech a distribuce databáze je prováděna pouze z master serveru a v nastavených časových intervalech bez ohledu na počet provedených změn. Tento způsob distribuce je závažným nedostatkem, neboť v intervalu mezi synchronizacemi může být obsah databáze na jednotlivých uzlech nekonzistentní. Autentizace uživatelů je prováděna za pomoci systému Kerberos. Autorizace je řešena pomocí ACL, které jsou realizovány jako součást metadat. Server je implementován jako sada programů pracujícím v uživatelském módu operačního systému, klient také z části pracuje v uživatelském režimu, ale s podporou vlastního modulu jádra operačního systému. Server je tvořen několika samostatnými procesy zajišťujícími autentizaci, autorizaci, poskytování dat či synchronizaci. Coda komunikuje na pevně daných portech přes TCP protokol, což umožňuje snadnou integraci s firewallem.

Na rozdíl od systému OpenAFS, Coda podporuje základní model multi-master replikace dat. Tento model zvyšuje dostupnost dat a dovoluje rozkládat zátež na více serverů. Neexistuje jediná zapisovatelná replika, ale je možné zapsat data na jakýkoli uzel. Multi-master replikace je implementována ve velmi omezeném rozsahu. Klient pokud zapisuje data, snaží se je zapsat paralelně na co nejvíce dostupných serverů, které obsahují podle interní databáze repliku zapisovaných dat. Už jen tento fakt je problematický pro použití na mobilních zařízeních, neboť výrazně zvyšuje požadavky na přenosové pásmo směrem od klienta, které je, jak bylo uvedeno v kapitole 2, omezené. Tento problém, byť v menší míře, se v Codě vyskytuje ještě jednou a to v případě, že jsou klientem požadována data. Při požadavku na data se klient snaží ověřit aktuálnost klientské cache a to tak, že kontaktuje co nejvíce dostupných serverů a ověřuje, zda některý ze serverů nemá aktuálnější data, což generuje další zatížení přenosové linky od klienta a zpoždění způsoběné čekáním na odpovědi.

Existují dva modely synchronizace replikovaných dat mezi servery. První je stejně, jako u OpenAFS, vyvolán proaktivně administrátorem. Druhý je vyvolaný požadavkem na replikovaná data, kdy při příchodu požadavku server, který požadavek obdržel, kontaktuje ostatní servery obsahující stejnou repliku dat a ověřuje aktuálnost své repliky dat. Pokud replika na poptávanén serveru není aktuální je zahájena aktualizace dat, což vede k dalšímu zpoždění odpovědi klientovi. Synchronizace replikovaných dat se navíc spouští už jen pouhým dotazem na metadata, například výpisem obsahu adresáře. To je potenciálně nebezpečné, neboť tohoto chování lze velice snadno zneužít k přetěžování systému, pokud by útočník kontaktoval jednotlivé servery a cyklicky procházel celé adresářové stromy.

Systém Coda se snaží lépe a hlouběji implementovat základní požadované vlastnosti distribuovaných souborových systému, které byly uvedeny v předchozích odstavcích, než je tomu u AFS. Stále se nedá hovořit o plnohodnotném distribuovaném souborovém systému, který by byl dobře nasaditelný pro mobilní

zařízení v geograficky rozsáhlých systémech. Primárním nedostatkem je systém replikací jak dat tak i metadat a systémové databáze. Další omezení vycházejí z histotické implementace, především se jedná o pevně dané přípojné body na serveru (*/vice*) i na klientech(*/coda*) či o omezení maximálního počtu souborů v jednom exportovaném svazku */vice* v rozmezí 256k-16M, kde toto rozmezí je nutné stanovit administrátorem při zakládání svazku. Tyto vlastnosti jsou pozůstatky z dlouhého vývoje. Již několikrát se uvažovalo o kompletním přepsáním Cody, ale zatím k tomu nikdy nedošlo. Přesto, že systém Coda je v mnoha ohledech návrhově propracovanější než starší systém AFS, Coda nikdy nebyla nasazena v tak masivním měřítku jakou systém AFS.

5.9.6 InterMezzo

InterMezzo[63] je distribuovaný souborový systém, který v současné době není příliš rozšířený, přesto jeho vlastnosti jsou velice zajímavé. Tento systém vychází ze systému Coda a byl navržen tak, aby byl co nejvíce modifikovatelný. V systému se nachází jeden server a libovolný počet klientů. Klienti přistupují k datům serveru s využitím vlastní lokální cache, která podporuje žurnálové operace. Klienti mohou pracovat ve dvou režimech. Jednak mohou mít omezenou velikost lokální cache a dočasně si uchovávají jen určitou množinu dat a druhak obsahují kompletní kopii datového skladu serveru. To umožňuje aby se za určitých podmínek stal klient serverem a tím převzal kontrolu. Systém byl primárně vyvíjen pro dva účely. Prvním možným použitím jsou servery pro sdílení dat s požadavkem vysoké dostupnosti. Druhým možným použitím je podpora off-line operací pro stanice, které nejsou trvale připojeny k sítí, avšak potřebují data ze serveru. V takovém případě se pracuje s lokální kopií dat a po opětovném připojení jsou data synchronizována a jsou zachytávány konflikty. InterMezzo je vyvinut pro spolupráci s žurnálovými souborovými systémy jako je XFS nebo JFS[64]. Systém umožňuje libovolně přejmenovávat sdílená data, stejně jako umožňuje přemapovávat datové zdroje na různé adresáře dle aktuální potřeby. Díky svým vlastnostem bývá někdy označován jako paralelní souborový systém. Toto označení vychází z principu replikace a uchování dat, neboť v tomto systému slouží server jako jedno z úložišť na stejné úrovni jako klienti, kteří s ohledem na svoji diskovou kapacitu, mohou obsahovat také všechna data. Pokud na klientovi dojde ke změně dat, je tato změna propagována na server, kde je zaznamenána a zároveň je okamžitě distribuována dalším klientům, kteří tak aktualizují svoji kopii datového skladu. Typickým využitím tohoto systému bylo nasazení jako podkladový souborový systém pro firmy WWW či FTP serverů, kde bylo požadavkem mít stejná data na více uzlech. Tento systém byl určen pro použití v operačním systému Linux. V současně době již není součástí aktuálních verzí jádra operačního systému a celý projekt se již několik let nevyvíjí.

5.9.7 Blue File System - BlueFS

BlueFS[65] je distribuovaný souborový systém, který byl již navrhován s ohledem na použití na mobilním zařízení. Jeho primárním cílem bylo snížit energetickou náročnost práce s distribuovaným souborovým systémem na mobilním zařízení. Svým základním návrhem vychází ze systému Coda. Na rozdíl od Cody je BlueFS

experimentální systém, který je implementován pouze pro operační systém Linux. Testy tohoto distribuovaného souborového systému a porovnání s další systémy (Coda, AFS) ukázaly, že se opravdu podařilo snížit energetickou náročnost daných operací. Základní myšlenky snižování energetické zátěže mobilních zařízení jsou dvě. Prvním je využití cache v podobě přídavných energeticky nenáročných paměťových medií jako je flash disk nebo usb disk. Druhým krokem je zavedení limitujících podmínek na distribuované operace a tím snížení čekání či nutnost opakovaného provádění operací při realizaci replikací. Na všechny operace je stanoven maximální časový rámec, který když se překročí, jsou ukončeny. Stejně tak nedokončené operace stanic, které se v průběhu vykonávání odpojí od serveru jsou okamžitě zrušeny. Ke kontaktování klientů jsou použity, stejně jako u AFS, *callbacky*, což znemožňuje fungování klientů na síťích, kde je použita technologie NAT, což je v rámci mobilních sítí velmi časté. Navíc jak bylo uvedeno v úvodu, tento experimentální souborový systém je aktuálně portován pouze na jeden operační systém.

5.9.8 Google File System

Google File System[66] byl vyvinut společností Google. Systém je optimalizován na ukládání velkého množství velkých datových souborů. Data jsou ukládána do bloků pevné velikosti 64MB. Tato pevná velikost umožňuje urychlit vyhledávání v rámci systému. Vyhledávací algoritmy jsou převzaté a upravené z webového vyhledávače Google. Základním cílem systému je umožnit uživatelům rychlý přístup k datům. Neřeší však příliš konzistenci těchto dat. Systém je tvořen jedním Master serverem a několika chunk servery. Master server je v rámci systému vždy jen jeden, obsahuje základní informace o uložených souborech a jejich mapování na data, správu oprávnění, uchovává a aktualizuje seznam jednotlivých chunk serverů. Chunk serverů je více a obsahují fyzicky uložená data. Základním požadavkem při návrhu tohoto systému bylo efektivní zpracování dat s velikostí nad 100MB. Menší soubory jsou také transparentně podporované, ale jejich zpracování není nijak optimalizováno. Mezi stěžejní vlastnosti tohoto systému patří optimalizace modifikace souboru zápisem na konec souboru, což je typickým požadavek u databází či logových souborů.

Integrita dat v celém systému je zajišťována pomocí kontrolních součtů, které kontroluje Master server. Zajištění integrity dat při současném zápisu je v tomto systému řešeno pomocí Master RW kopie. V systému replikací je vždy jen jeden zdroj určen k zápisu. Která replika dat bude určena pro zápis určuje Master server. V rámci systému může reálně dojít k poškození dat, neboť se replikují mezi jednotlivými chunk servery bez zásahu Master serveru. Problém nastane, pokud Master server včas neodhalí chybu v kontrolním součtu a data se zreplikují na všechny chunk servery. Pokud je problém odhalen včas a existuje alespoň jedna správná kopie dat, jsou z ní data obnovena na všech chunk serverech. Informace o mapování jmén souborů na datové bloky na chunk serverech jsou na straně klienta cachována, takže určitou dobu může klient fungovat bez nutnosti komunikace s Master serverem.

Tento systém je optimalizován na málo se měnící data či na data, jejich modifikace je realizována zápisem na konec souboru.

5.10 Zhodnocení existujících řešení

Z analýzy současných distribuovaných souborových systémů plyne, že pro použití na mobilních zařízeních a v geograficky rozsáhlých systémech, nejsou vhodné. Důvody shrneme v následujích dvou podkapitolách.

5.10.1 Nedostatky z pohledu mobilních zařízení

Hlavním nedostatkem současných distribuovaných souborových systémů při použití na mobilních zařízeních je závislost na počítačové síti. V případě zhoršené kvality přenosu dat, může docházet při přístupu k distribuovaným datům k výraznému zpomalení nebo zahlcení systému a to hlavně v případech systémů bez lokální cache, kde se tento problém projeví mnohem výrazněji. Typickou ukázkou je NFS. Při použití NFS na lince se ztrátovostí paketů, dochází velice často k zamrznutí klientské stanice, neboť požadavky jsou posílány několikrát a v systému se hromadí procesy požadující odpověď od NFS. Navíc v případě, že klient několikrát nedostane odpověď od serveru v daném časovém limitu, restartuje spojení, což vede k dalším prodlevám při novém navazování spojení. Po delší době tak dojde k trvalému uvíznutí služeb, neboť I/O operace jsou blokující.

Dalším problémem některých řešení je, že nepoužívají lokální cache. Lokální cache je pro fungování na mobilním zařízení zcela zásadní, neboť umožňuje částečně eliminovat nestabilitu sítě, především při opakované práci se stejnými daty. Navíc šetří datové přenosy a tím i energii mobilního zařízení.

Existující řešení, popsaná v předchozích kapitolách, neumožňují žádným způsobem dynamicky měnit prioritu serverů a tím optimalizovat použitou datovou cestu nebo není, jako například u AFS, možné tyto priority měnit dynamicky za chodu systému s ohledem na aktuální stav datových sítí. Tento nedostatek je velmi závažný, neboť možnost volby optimálního zdroje může výrazně zvýšit přenosovou rychlost a snížit tak celkovou dobu přenosu, čímž by se jednak zvýšila rychlost systému z pohledu klienta, druhak by se dále zmenšila energetická náročnost díky kratší délce realizovaného přenosu.

Velkým nedostatek je také omezená či neexistující možnost využívat pro zápis libovolný server souborového systému. Některé systémy nepodporují replikaci vůbec, u většiny je k dispozici pouze systém s Master RW kopií, které v systému zavádí princip úzkého hrdla. Navíc, aby mohl být požadavek na dynamické směrování požadavků z předchozího odstavce realizovatelný, musí v systému existovat více rovnocenných zdrojů, aby bylo mezi čím vybírat. Souhrnné porovnání vlastností existujících řešení popsaných v předchozích kapitolách a nového distribuovaného souborového systému KIVFS je v tabulce 5.3.

5.10.2 Nedostatky z pohledu geograficky rozsáhlých datových sítí

Distribuovaný souborový systém zprostředkovává data operačnímu systému a následně uživatelům. Z pohledu operačního systému se jedná o I/O operace, které jsou v jádře operačního systému blokující. Pokud proces potřebuje přistoupit k datům na souborovém systému požádá o tyto data jádro operačního systému a to přes univerzální vrstvu VFS zavolá obsluhu konkrétním souborovým systémem.

Pokud tímto souborovým systémem je distribuovaný souborový systém je situace o to složitější, že se k datům nepřistupuje jen v rámci lokálního pevného disku, ale prostřednictvím počítačové sítě. Základní úskalí tohoto přístupu byla popsána v kapitole 2 s ohledem na vlastnosti mobilních zařízení. Ale i na metalických či optických spojích mezi datovými centry či servery nástává problém související s distribuovaným přístupem. Datová centra či servery jsou typicky propojena optickými nebo metalickými linkami s vysokou přenosovou rychlosti v řádu desítek až stovek Gbps. Problém s rychlostí a propustností datového spojení, jako u mobilních zařízení, zde nenastává. Problémem je geografická odlehlost datových center a s ní spojené zpoždění RTT. Toto zpoždění je zapříčiněno použitou přenosovou technologií a fyzikálními možnostmi šíření signálu pro dané přenosové médium, délkou spoje a použitou technologií. U kontinuálních přenosů velkých souborů není RTT zpoždění zásadní problém, ale v rámci distribuovaného souborového systému nerozhoduje o požadované operaci každý uzel sám, ale typicky se o pořadí a platnosti operací hlasuje pomocí zpráv mezi jednotlivými uzly. To znamená, že kromě samotného přenosu souboru se v distribuovaném souborovém systému přenáší velké množství velmi malých zpráv, sloužících pro řízení chodu systému. V takovém případě toto zpoždění negativně ovlivňuje chování jednak distribuovaného souborového systému, neboť jednotlivé uzly musejí čekat na zprávy od dalších uzlů, druhak ovlivňuje chod operačního systému klientské stanice, neboť během čekání na odpověd od dalších uzlů je proces požadující data blokován v jádře operačního systému při čekání na dokončení I/O operace. Tyto procesy jsou z pohledu operačního systému neukončitelné a blokující, což je jednak nepříjemné pro uživatele, druhak vede k přetížení systému, neboť se v něm budou zpožďovat a hromadit nezpracované zprávy.

Název	**Replikace**	**RW replikace**	**Online replikace**	**Replikace částí dat**	**Podpora NAT**	**Šifrování dat**	**Dyn. směrování**
NFS	Ano	Ne	Ne	Ne	Ano	Ne	Ne
OpenAFS	Ano	Ne	Ne	Ne	Ne	Ne	Ne
Coda	Ano	Ano	Ano	Ne	Ne	Ne	Ne
InterMezzo	Ano	Ano	Ano	Ne	Ano	Ne	Ne
SFS	Ne	Ne	Ne	Ne	Ano	Ne	Ne
DFS	Ne	Ne	Ne	Ne	Ano	Ne	Ne
SMB	Ne	Ne	Ne	Ne	Ano	Ne	Ne
CiFS	Ne	Ne	Ne	Ne	Ano	Ne	Ne
BlueFS	Ano	Ne	Ne	Ne	Ne	Ne	Ne
Google file system	Ano	Ano	Ano	Ne	Ano	Ne	Ne

Tabulka 5.3: Souhrnné porovnání vlastností popsaných distribuovaných souborových systémů

6. Cíle práce

Práce se zabývá problémem přístupu mobilních zařízení k rozsáhlým datovým skladům a jejich sdílení. Po prostudování možností mobilních zařízení se ukázalo, že vhodný způsob řešení tohoto problému je použití distribuovaného souborového systému. S ohledem na zjištěné možnosti mobilních zařízení popsaných v kapitole 2, dále pak v souvislosti s vlastnostmi distribuovaných souborových systémů a jejich existujících řešení popsaných v kapitole 5 a následně shrnuté v tabulce 5.3, jsme zjistili, že žádné existující řešení zcela výše uvedený problém neřeší.

Základní problém spojení těchto dvou technologií je síťové fungování obou systémů [67]. Mobilní zařízení často nedisponuje kvalitním a stálým připojením k počítačové síti, proto je nutné data stahovat ze zdroje, který je dokáže poskytnout v co nejkratším čase, aby se minimalizovala doba přenosu a tím i možnost vzniku chyby. Splnění tohoto požadavku předpokládá v distribuovaném souborovém systému existenci více zdrojů poskytujících replikovaná data a to jak pro čtení tak pro zápis tak, aby bylo možné kontaktovat nejvýhodnější server z pohledu síťové odezvy. Takové řešení v současné době neexistuje.

Aby libovolné mobilní zařízení, nacházející se kdekoli na světě, mohlo efektivně vybírat z více datových zdrojů, je nutné, aby replikovaná data byla rozmístěna po celém světě, což předpokládá existenci geograficky rozsáhlého systému. Základním nedostatkem geograficky rozsáhlých systémů je RTT zpoždění, které bylo popsané v kapitole 3. Řešením RTT zpoždění je vhodný výběr datových cest. Výběr vhodné datové cesty je v počítačových sítích obvykle realizován pomocí některého z dynamických směrovacích protokolů popsaných v kapitole 4. Nasazení dynamického směrovacího protokolu předpokládá možnost konfigurace všech směrovačů v počítačové síti používané distribuovaným souborovým systémem. Tento požadavek je pro geograficky rozsáhlé systémy typicky nereálný. Směrování požadavků bude nutné řešit bez možnosti přístupu k fyzickým směrovačům. Z uvedených informací vyplývají následující cíle:

1. Navrhnout a implementovat nový souborový systém. V tomto novém experimentálním distribuovaném souborovém systému, nazvaném KIVFS, bude umožněno efektivní propojení mobilních zařízení a geograficky rozsáhlých datových uložišť.

2. Navrhnout pro KIVFS systém replikací umožňující nejen číst, ale i zapisovat na libovolnou repliku v rámci distribuovaného souborového systému a systém sám musí být schopen provést asynchronní replikaci dat na další uzly bez toho, aby uživatel musel čekat až se dokončí replikace dat na všechny uzly. Tento mechanizmus replikací budeme nazývat Multi-Master RW On-line replikace.

3. S ohledem na možnost fungování KIVFS v geograficky rozsáhlých systémech, navrhnout systém efektivní volby datové cesty na aplikační úrovni.

7. KIVFS

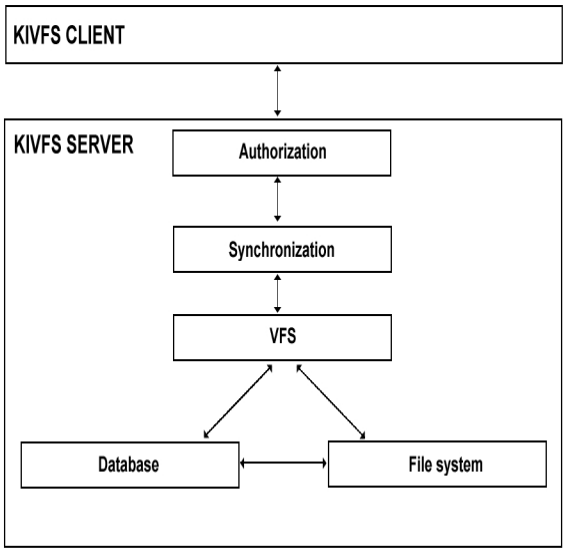
Principy a vlastnosti popsané v předchozích kapitolách jsou obecně platné v libovolném distribuovaném souborovém systému. Po prozkoumání existujících systémů jako je OpenAFS, Coda, NFS4 i systémů právě vznikajících jako je BlueFS bylo zjištěno, že pro realizace cílů stanovených v kapitole 6 není žádné z existujících řešení vhodné z důvodů, které jsou uvedeny u popisu jednotlivých řešení. Pro realizaci vytyčených cílů byl navržen nový distribuovaný souborový systém, který není zatížen návrhovou ani implementační historií, která by řešení omezovala. Distribuovaný souborový systém KIVFS je stěžejní pro realizaci dříve vytyčených cílů.

7.1 Základní architektura

Distribuovaný souborový systém KIVFS je založen na architektuře klient - server.

Realizace klienta se pro různé platformy liší, detaily jednotlivých implementací klientů budou popsány v kapitole 7.8.

Server je rozdělen na pět modulů, které jsou znázorněny na obrázku 7.1.



Obrázek 7.1: Model základní architektury KIVFS

Funkce a principy práce jednotlivých modulů budou detailně popsány v následujících kapitolách, v tabulce 7.1 je uvedena základní funkce jednotlivých modulů.

Jednotlivé moduly spolu komunikuji pomocí vlastního KIVFS protokolu, který popisuje kapitola 7.2. Prvním důvodem tohoto dělení je skutečnost, že rozdělení umožňuje, aby každý modul mohl pro jednotlivé uzly fungovat na samostatném hardwaru. Druhým důvodem rozdělení systému na více modulů byla skutečnost, že KIVFS byl vyvinut a navržen jako experimentální systém, v rámci kterého je požadována pro jednotlivé experimenty, možnost snadné výměny komponent. Všechny moduly patřící k jednomu uzlu mohou fungovat společně na jediném

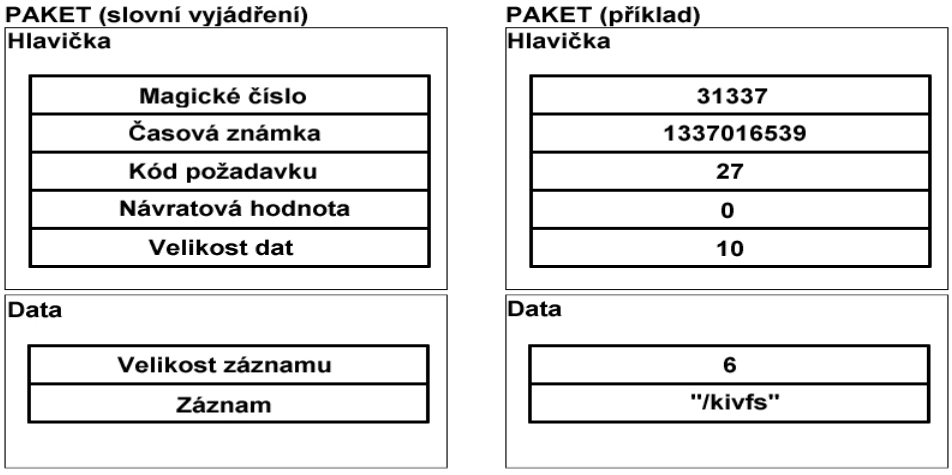
Název modulu	Základní funkce
Auth	Vstupní brána do systému, řeší šifrování spojení, autentizaci a autorizaci, detailně bude popsán v kapitole 7.3
Sync	Řeší synchronizaci požadavků a optimalizaci datových cest a jedná se o stěžejní modul této práce, detailně bude popsán v kapitole 7.4
VFS	Zajišťuje jednotný přístup k datům a metadatům, dle operací rozděluje požadavky mezi DB a FS moduly, detailně bude popsán v kapitole 7.5
DB	Zajišťuje komunikaci s databází, obsluhuje požadavky na metadata a podpůrná data uložená v databázi, detailně bude popsán v kapitole 7.6
FS	Zajišťuje operace s obsahem datových souborů, jejich správu, replikaci a deduplikaci, detailně bude popsán v kapitole 7.7

Tabulka 7.1: Základní funkční význam jednotlivých modulů KIVFS serveru

serveru nebo mohou fungovat odděleně a to jak v rámci lokální sítě, tak i prostřednictvím veřejného internetu. Navržená architektura navíc umožňuje snadné škálování do šířky.

7.2 Komunikační protokol KIVFS

Jednotlivé moduly serveru mezi sebou, stejně jako klient a server, komunikují pomocí vlastního protokolu, který je navržen tak, aby nepřenášel žádná nadbytečná data a umožňoval pracovat s minimálními datovými přenosy. Struktura zprávy zasílané KIVFS protokolem je znázorněna na obrázku 7.2. Jak je z obrázku patr-



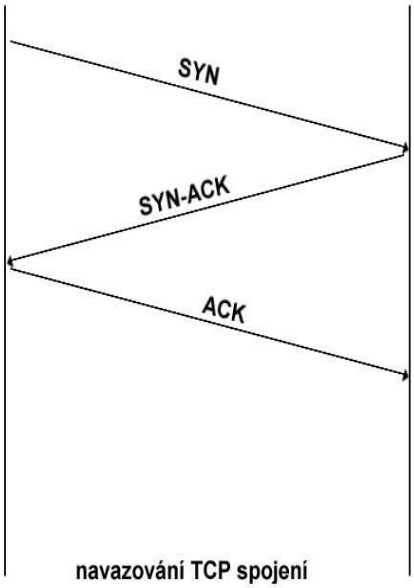
Obrázek 7.2: Zpráva KIVFS protokolu

né, zpráva je dělena na dvě části. První, označená jako hlavička, je vždy povinná a obsahuje nejnutnější informace k tomu, aby zprávu bylo možné zpracovat. Druhá část zprávy, označená jako data, je nepovinná a je použita jen pro ta volání, kde jsou nějaká data předávána. Protokol je dostatečně obecný nato, abychom s jeho pomocí pokryli veškerou komunikaci jak mezi klientem a serverem tak i mezi servery samotnými. Jednotný formát zprávy je výhodný i z pohledu rychlosti zpracování, což je podstatné především pro mobilní zařízení. Detailní popis jednotlivých hodnot ve zprávě je možné najít v [68].

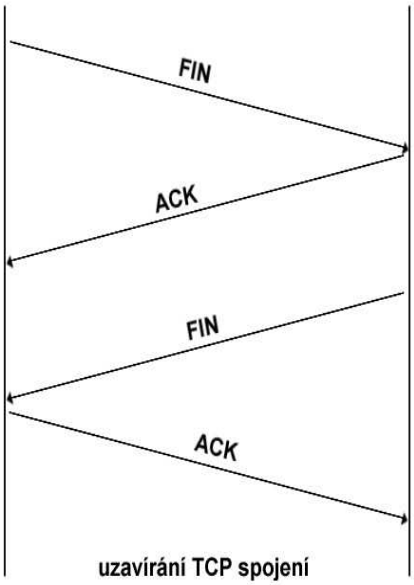
7.2.1 Minimalizace síťových zpoždění

Protokol KIVFS, který je popsán v předchozí kapitole 7.2, je postaven nad přenosovým protokolem TCP a je implementován pomocí *Berkley socketů* typu Inet. Přenosový protokol TCP se na rozdíl od druhého běžně používaného protokolu UDP vyznačuje několika vlastnostmi, které jsou pro fungování distribuovaného souborového systému výhodné. Především se jedná o následující vlastnoti:

- Udržování navázaného spojení: v každém okamžiku víme, zda je spojení ještě aktivní či nikoliv, neboť spojení je na svém počátku navázané, jak ukazuje obrázek 7.3 a při zániku aktivně ukončované, jak ukazuje obrázek 7.4.



Obrázek 7.3: TCP: Navazování spojení



Obrázek 7.4: TCP: Ukončování spojení

- Retransmit paketů: pokud do stanoveného času nedojde k potvrzení paketu, je odeslán znovu, bez nutnosti tuto situaci řešit na vyšších vrstvách ISO/OSI modulu. Řešení na vyšších vstvách ISO/OSI je vždy náročnější na dostupné zdroje, především více zatěžuje procesor.
- Uspořádání paketů: pokud se stane, že pakety dorazí v jiném než sekvenčním pořadí, TCP protokol si umí jejich pořadí před zpracováním uspořádat a tak eliminovat nutnost retransmitu, který by způsobil zpoždění komunikace.

Tyto vlastnosti, které u UDP protokolu nenajdeme, jsou zaplacené vyšší přenosovou řežií, která se projevuje nejvíce na začátku a na konci komunikace, jak vidíme na obrázcích 7.3 a 7.4, ale i při jejím průběhu kvůli potvrzování doručení jednotlivých paketů. Tyto nedostatky jsou výraznější, pokud je typickým komunikačním modelem přenášení velkého množství malých paketů. Tato situace v distribuovaných systémech nastává při synchronizaci logických hodin. Jelikož je distribuovaný systém na fungování logických hodin závislý, může docházet k výraznému zpomalení celého systému. V KIVFS se tomuto problému předchází udržováním trvale otevřených spojení, což je možné realizovat, neboť počet uzlů, které spolu trvale komunikují, je v čase konstantní. Při startu systému se naváže spojení všech vrstev mezi sebou a zároveň vzájemné spojení všech serverů a toto spojení se po celou dobu chodu systému udržuje otevřené. Spojení jsou kontrolována a pokud nějaké, například z důvodu síťového výpadku, selže je okamžitě znovu navázáno. Tímto způsobem eliminujeme počet operací otevírání a ukončování spojení na minimum.

7.3 Auth modul

Jediným vstupním bodem do serverové části systému KIVFS je Auth modul. Auth modul má několik úkolů v KIVFS architektuře. Základním úkolem Auth modulu je fungování jako proxy server pro další moduly. To je výhodné řešení, neboť základní operace zabezpečení, kterými jsou šifrování komunikace, autentizace, autorizace a logování, je možné realizovat v jediném bodě. Pokud požadavek klienta projde tímto jediným vstupním bodem, můžeme jej pro operace na dalších modulech považovat za legitimní a zabezpečený. Auth modul neřeší právo přístupu k datům samotným, tedy práva na úrovni souborového systému. O tuto kontrolu se stará až FS vrstva, která bude podrobněji popsána v kapitole 7.5.

Veškerá komunikace s distribuovaným souborovým systémem by měla být šifrovaná. Jelikož KIVFS je experimentální systém, který kromě poskytování dat slouží i jako nástroj testování řešení různých problémů v rámci distribuovaných systémů, je možné princip šifrování měnit v rámci konfigurace.

Pro šifrování je v KIVFS zvolena symetrická šifra AES. Tato volba vychází z testů provedených v rámci práce [71], v rámci které proběhlo testování parametrů šifrování z pohledu bezpečnosti, rychlosti a zatížení procesoru. Poslední parametr je velice zásadní s ohledem na požadavek použití na mobilních zařízeních, které disponují omezenými výkonovými parametry, jak bylo uvedeno v kapitole 2. Dalším důvodem volby procesorově úsporného řešení je i skutečnost, že kromě výkonu procesoru je dalším limitujícím faktorem výdrž baterie, neboť výpočetně složitější operace jsou vždy energeticky náročnější. Výsledky provedených testů ukazuje následující tabulka 7.2.

Přesto, že z testování vyšlo nejlépe šifrování RC4-MD5 a RC4-SHA, byla pro použití v KIVFS z důvodu vyššího zabezpečení zvolena varianta AES256-SHA.

Po úspěšném navázání komunikace (šifrované nebo nešifrované podle nastavení), je jako první operace vyžadováno ověření uživatele. Uživatelé jsou ověřováni dvoufázově. K autentizaci se používá systém Kerberos, mezi jehož hlavní přednosti patří skutečnost, že heslo není nikdy v žádné ani šifrované formě, přenášeno prostřednictvím počítačové sítě. Heslo je zadáno lokálně na klientovi a následně je s ním a dalšími parametry jako jsou aktuální čas, jméno počítače, doména ve

Šifra	**CPU klient[s]**	**CPU server[s]**	**Rychlost [MB/s]**
DES-CBC-SHA	8.71	9.39	19.12
DES-CBC3-SHA	19.55	19.70	9.38
RC4-MD5	2.61	4.16	39.22
RC4-SHA	2.88	4.60	38.20
AES128-SHA	4.25	6.14	30.37
AES256-SHA	4.63	7.08	26.02

Tabulka 7.2: Porovnání využití CPU na klientovi a serveru a přenosové rychlosti, při použití různých šifrování

které se nachází a další, provedena matematická operace a teprve její výsledek je přenášen po síti. Obdobná operace, ale už jen s výsledkem operací provedených na klientovi, je provedena na straně serveru a na základě ověření platnosti je vystaven ticket, který je předán zpět klientovi, a který je možné následně použít k ověření vůči dalším částem aplikace, což se v KIVFS využívá k autentizaci přístupu mezi jednotlivými moduly. Využití systému Kerberos má ještě jeden praktický důvod a tím je skutečnost, že Kerberos je součástí Active Directory[80], který je primárním autentizačním a autorizačním nástrojem pro systémy Microsoft Windows ve větších organizacích a tímto propojením získáme snadnou možnost integrace KIVFS do již existujícího prostředí. Autentizace KIVFS, podobně jako je tomu například u OpenAFS, je svázána s jedním konkrétním realmem v systému Kerberos. Tento realm je nastaven v konfiguračním souboru a je společný pro celý KIVFS.

U velkých organizací, jako jsou univerzity, může být v rámci jednoho Kerberos realmu až tisíce nebo desítky tisíc uživatelů. To není pro fungování KIVFS nijak limitující, ale nemusí být vždy žádoucí, aby všichni uživatelé v rámci jednoho realmu měli přístup ke zdrojům poskytovaným distribuovaným souborovým systémem. Tato selekce je, v první fázi, realizována pomocí autorizace. Autorizace v KIVFS probíhá dvoufázově, jednak je ověřováno zda uživatel má právo se k systému připojit. Tuto část autorizace řeší Auth vrstva. Druhá fáze autorizace, v rámci které se určuje oprávnění uživatele manipulovat s daty, je řešena na DB modulu při procházení metadat. Autentizace určuje, zda uživatel má právo se připojit k systému, je selekcí vybraných uživatelů z množiny uživatelů autentizovaných pomocí systému Kerberos. Tato selekce je v KIVFS prováděna na základě seznamu uživatelů, jejichž uživatelská jména jsou uložena v databázi. Auth modul se v tomto kroku dotazuje DB modulu, zda daný uživatel, který se pokouší připojit k systému, je uveden v seznamu oprávněných uživatelů. Detaily uložení seznamu uživatelů budou popsány spolu s DB modulem v kapitole 7.6. Základní princip je takový, že součástí databáze je tabulka *USERS*, kde jsou vyjmenováni všichni uživatelé s právem přistoupit do KIVFS. Dotaz do databáze není realizován přímo připojením do databáze a pomocí SQL dotazu, ale pomocí interní komunikace Auth modulu s DB modulem prostřednictvím KIVFS protokolu. Tímto způsobem je odstíněna vazba na konkrétní implementaci databáze.

Pokud se autentizace či autorizace nepodaří, je spojení s klientem okamžitě ukončeno a o tomto pokusu o připojení a důvodu zamítnutí je proveden záznam do logu. K logování je využívána služba syslog[81], což je běžná praxe v Unixových

systémech.

V případě úspěšné autentizace a autorizace je navázáno spojení se Sync modulem, kterému jsou předávány veškeré požadavky klienta až do doby ukončení spojení.

Kromě bezpečností funkce Auth modul umožňuje realizovat proxy server pro další platformy, kde není snadné implementovat podporu nativního KIVFS protokolu. Jedná se například o mobilní zařízení s operačním systémem Android. Na této platformě se nativně předpokládá komunikace pomocí volání API jako jsou JSON[82] či REST-API[83], které je zapouzdřené v HTTP[84],[85] nebo HTTPS[86] protokolu. Základní Auth modul je možné nahradit za modifikovanou verzi, která bude realizovat proxy server mezi KIVFS protokolem pro další moduly KIVFS a libovolným protokolem, například HTTP, pro klienty. Příkladem implementace modifikovaného Auth modulu je práce [74], v rámci které byla realizovaná proxy právě pro prokol HTTPS a zařízení Android.

7.4 Sync modul

Sync modul v KIVFS představuje synchronizační bod celého systému. V případě KIVFS, jakožto decentralizovaného systému s velkým množstvím uzlů, které jsou navíc geograficky oddělené, je výhodné k synchronizaci požadavků použít logických hodin, jejichž vlastnosti a princip fungování je popsán v kapitole 5.5.2. Konkrétně byla vybrána varianta Lamportových skalárních hodin s úplným uspořádáním. Důvodem této volby byla skutečnost, že při použitích tohoto druhu logických hodin bude přenášeno nejmenší množství dat. Tento argument může na první pohled vypadat nesmyslně, neboť se v případě přenosu logických hodin nejedná o přenos velkého množství dat v rámci jedné zprávy či operace. Je ale třeba si uvědomit, že v rámci distribuovaného souborového systému je nutné zajistit jednotné pořadí všech operací, v našem případě navíc rozšířené o synchronizační požadavky DB modulu. Přenášené zprávy obsahující logické hodiny jsou malé objemem, ale jejich počet je obrovský. Tento počet bude narůstat s narůstajícím počtem uzlů. Porovnání přenesených dat pro skalární, vektorové a maticové hodiny v prostředí s různým počtem uzlů ukazuje následující tabulka 7.3.

Počet uzlů	**Skalární [B]**	**Vektorové [B]**	**Maticové [B]**
5 uzlů	4	20	100
10 uzlů	4	40	200
100 uzlů	4	400	40000

Tabulka 7.3: Porovnání přenášených dat pro skalární, vektorové a maticové logické hodiny

Jak je z tabulky patrné, pro prostředí se 100 uzly je varianta skalárních logických hodin 10.000x úspornější než varianta maticových logických hodin. Tento rozdíl se ještě výrazněji projeví v případě, že nebudeme uvažovat o přenosu jedné samostatné zprávy, což je nereálná situace. V reálném provozu je pro úplné uspořádání skalárních lamportových hodin, vyslána zpráva s návrhem časové značky všem uzlům systému, každý z uzlů odpovídá svým protinávrhem, následně je

všem odeslána zpráva s informací o zvolené časové značce. Úspora námi zvoleného řešení je v tomto případě ještě výraznější, jak ukazuje tabulka 7.3.

Počet uzlů	**Skalární [B]**	**Vektorové [B]**	**Maticové [B]**
5 uzlů	60	300	1500
10 uzlů	60	600	3000
100 uzlů	60	6000	600000

Tabulka 7.4: Porovnání přenášených dat pro skalární, vektorové a maticové logické hodiny pro jednu synchronizační operaci

Samozřejmě kromě velikosti přenášených dat narůstá pro vektorové i maticové hodiny i časová a paměťová náročnost zpracování požadavků, neboť místo prostého porovnání dvou čísel je nutné porovnávat vektory nebo dokonce matice.

7.4.1 Sychronizace požadavků

Sync modul přebírá požadavky klienta od Auth modulu a dále požadavky od interních částí systému, které potřebují zajistit jednotné pořadí operací v celém distribuovaném systému. Sync modul zajišťuje synchronizaci dvojího druhu požadavků:

- Požadavky od klienta, tedy požadavky na provedení operací nad distribuovaným souborovým systémem. Jedná se o operace výpisu obsahu adresáře, tvorba, mazání a modifikace adresářů a souborů.
- Požadavky na databázový server. Jak bude následně popsáno v kapitole 7.6, každý jednotlivý uzel KIVFS má vlastní kopii celé databáze. KIVFS není pevně svázáno s žádným konkrétním databázový systémem. Jednotlivé databázové systémy nemusí vždy podporovat replikace SQL operací a nebo neumožňuje synchronní replikace SQL operací v reálném čase, proto bylo navrženo vlastní, na databázovém systému zcela nezávislé, řešení replikace SQL operací.

Pro každou uvedenou skupinu požadavků, které jsou na sobě vzájemně nezávislé, jsou v KIVFS implementovány vlastní logické hodiny.

Pro zajištění postupného zpracování požadavků na synchronizaci jsou v KIVFS implementovány dvě dvojice nezávislých front:

- Fronta async-client: Nezpracované požadavky od klientů čekající na přidělení časové značky.
- Fronta sync-client : Požadavky od klientů, které už jsou seřazené na základě přidělené časové značky a čekající na zpracování.
- Fronta async-db: Nezpracované databázové požadavky čekající na přidělení časové značky.
- Fronta sync-client : Databázové požadavky, které už jsou seřazené na základě přidělené časové značky a čekající na zpracování.

Každou frontu zpracovává samostatné vlákno Sync modulu, aby nedocházelo k vzájemnému ovlivňování. Na systémech s více jak jedním jádrem CPU, což je dnes drtivá většina dostupných zařízení, nám toto rozdělení umožňuje realizovat paralelní zpracování požadavků. To je velmi důležité, neboť Sync modulem prochází veškerá komunikace a proto má tento modul klíčový význam pro celkovou výkonnost systému.

Požadavky jsou Sync modulu doručeny ze dvou zdrojů. Prvním zdrojem požadavků je klient. Klientský požadavek je Sync modulu doručen od Auth modulu. Druhým zdrojem požadavků je DB modul. Sync modul po přijetí požadavku od Auth nebo DB modulu zařadí tento požadavek do fronty nesynchronizovaných požadavků async-client nebo async-db, podle zdroje požadavku. Tyto fronty jsou zpracovávány podle algoritmu FIFO. První požadavek, který je vložen do fronty je také jako první z fronty vyjmut a zpracován. Pod pojmem zpracování se v případě požadavků zařazených v async-client a async-db frontě myslí synchronizace těchto požadavků pomocí Lamportových skalárních hodin s výsledným přiřazením časových značek T_{cli} a T_{db}.

Prvním krokem synchronizace pořadí požadavků je odeslání návrhu na časovou značku T_{tmp}. Tato časová značka T_{tmp} je $T_{tmp} = T + d$, kde T je poslední hodnota lokálních skalárních hodin T_{cli} nebo T_{db}, podle fronty ze které byl požadavek vybrán. Tato časová značka T_{tmp} je následně odeslána všem ostatním uzlům, jako návrh nové hodnoty logických hodin. Uzel odesílající návrh nové hodnoty logických hodin musí znát všechny ostatní uzly patřící do jednoho systému. Jednotlivé uzly jsou v KIVFS adresovány pomocí IP adres. Seznam IP adres všech uzlů je znám z konfigurace systému, kde musí být všechny uzly specifikovány. Příklad takové konfigurace jednoho uzlu je znázorněn ve výpisu konfigurace 7.1.

Výpis 7.1: Ukázka konfigurace pro sync servery

```
[sync]
ip=147.228.67.121
port=30002
vfsport=30003
syncportcli=30010
syncportdb=30011
routeport=30012
recoveryport=30013
transactionport=30014
routesyncport=30015
local=node1
node1=147.228.67.121
node2=147.228.67.122
node3=147.228.67.123
node4=147.228.67.124
```

Porty jednotlivých modulů jsou na všech uzlech nastavené pro jednoduchost stejně, což zjednodušuje konfiguraci firewallu. Porty, na které jsou odeslány nabídky hodnoty časových značek, jsou :

- *syncportcli*: pro synchronizaci požadavků od klientů a nastavení časové

značky T_{cli}

- *syncportdb*: pro synchronizaci požadavků od databázové vrsty a nastavení časové značky T_{db}

.

Zpráva je odeslána pomocí KIVFS protokolu. Struktura zprávy je stejná v případě synchronizace požadavku od klienta, jako v případě synchronizace požadavků od databázové vrstvy. Uzel, který přijme zprávu s návrhem časové značky T_{msgIn} tuto značku porovná se svými lokálními hodinami T. Na základě porovnání určí časovou značku, která bude odeslána zpět uzlu, který žádal o synchronizaci. Možné stavy porovnání jsou :

- Pokud $T_{msgIn} > T => T_{msgOut} = T_{msgIn}$
- Pokud $T_{msgIn} \leq T => T_{msgOut} = T + d$

Jak bylo uvedeno v kapitole 5.5.2, je d, konstanta, která je dohodnutá v konkrétním distribuovaném souborovém systému a jedná se o kladné celé číslo různé od nuly. V KIVFS je tato konstanta d rovna jedné.

Když se uzlu, který žádal o synchronizaci, vrátí zprávy od ostatních uzlů, provede se vyhodnocení hodnot navržených časových značet $Tmsg_i$, kde i je počet uzlů mínus jedna. Hodnotu časové značky T určíme jako $T = max(Tmsg_i, T_{msg})$. Po určení T je informace o vybrané hodnotě odeslána na všechny uzly. Jednotlivé uzly přijmou zprávu T_{msg} a pomocí ní aktualizují své lokální hodiny T.

Akce, která následuje po synchronizaci hodin pro jednu operaci, se zásadně liší pro případ synchronizace datových požadavků od klientů či ostatních serverů a pro synchronizaci požadavků databázové vrstvy. Jednotlivé datové operace se vykonávají jen na uzlu, který žádal o synchronizaci operace (a to i v případě, že se jedná o zápis do dat, která mají v rámci systému zavedenou repliku, jak bude popsáno v kapitole 7.7.5). Po odeslání zprávy s dohodnutou časovou značkou pro danou operaci, je tato přesunuta z front *asynx-** do front *sync-** a následně je operace vykonána.

7.4.2 Vykonání a archivace požadavků

Po té, co je požadavek označen časovou značkou a přeřazen do jedné z front *sync-**, je připraven ke zpracování. Zpracování požadavku probíhá v samostatném vlákně tak, aby mohla paralelně probíhat synchronizace časových značek a další operace jako hledání optimální cesty. Zpracování požadavku je z pohledu Sync modulu snadné, jedná se jen o předání požadavku dalšímu modulu a to :

- Předání *VFS modulu* - v případě, že se jedná o datový požadavek
- Předání *DB modulu* - v případě, že se jedná o databázový požadavek

Operace požadované a následně vykonávané na DB modulu se pro zajištění vlastní replikace databáze pomocí KIVFS musejí provést na všech uzlech a samozřejmě ve stejném pořadí. Pokud se odesílá odpověď s nově vybranou časovou značkou pro danou operaci od inicializačního uzlu, je tato zpráva doplněna i o požadovanou operaci včetně potřebných dat. Po synchronizaci databázového

požadavku se na uzlech, které nejsou iniciátorem operace, provede instalace požadavku do fronty *sync-db*. Tato fronta slouží jako transakční log databázových operací. Operace požadovaná klientem je také provedena, ale jen na uzlu, na kterém operace vznikla a informace o provedení a případně požadovaná data jsou prostřednictvím Auth modulu předány zpět klientovi.

Po obdržení návratové informace z FS případně DB modulu je požadavek z fronty *sync-** přesunut do fronty *archiv-**. Tvorba front *archiv-** má dvojí význam. Prvním důvodem je možnost zpětného dohledání posloupnosti operací, neboť jednotlivé fronty tvoří transakční log. Druhým důvodem je možnost provedení *recovery* operací na odpojeném uzlu, jak bude popsáno dále.

7.4.3 Výpadek uzlu a jeho obnova

V distribuovaných systémech a v geograficky rozsáhlých systémech zvláště, nastávají situace, kdy jeden nebo více uzlů nejsou pro ostatní uzly dostupné. Existují dva důvody vzniku této situace:

- Výpadek síťové konektivity
- HW nebo SW výpadek uzlu či jeho části

V obou výše zmíněných případech vzniká problém, neboť k danému uzlu se nedostávají všechny nebo některé zprávy a tedy se nemůže účastnit hlasování při přiřazování časových značek operacím a zároveň se u něj neaktualizuje lokální databáze, neboť spolu se synchronizačními zprávami nepřicházejí ani požadavky na aktualizaci databáze. Z důvodu možnosti vzniku dočasně nedostupných uzlů, musíme v systému vyřešit následující nově vzniklé situace:

- *K uzlu, který je nedostupný pro část nebo celý zbytek systému, mohou být připojeni uživatelé, jejichž požadavky nebudou moci být synchronizovány v celém systému.* Tato situace je KIVFS vyřešena už přímo v návrhu chování systému. Každý požadavek od klienta se nejprve musí označit časovou značkou. O přidělení časová značky je v systému hlasováno všemi uzly. Tím, že částečně dostupný uzel není schopen komunikovat s částí nebo všemi ostatními uzly, nedojde k synchronizaci časových značek a požadavek skončí chybou, která signalizuje, že byl překročen časový rámec vyhrazený pro provedení jedné operace. Tento časový limit je nastaven v konfiguračním souboru direktivou *Timeout* a jeho výchozí hodnota je nastavena 30s. Hodnota 30s byla zvolena s ohledem na výchozí hodnotu timeoutu TCP/IP spojení pro Unixové systémy. Delší čekání na odpověd od klienta by, ve výchozím stavu konfigurace TCP/IP, nedávalo smysl, neboť požadavek by skončil timeoutem síťové komunikace. Díky nutnosti přidělení časové značky nemůže nastat situace, kdy by jeden dočasně nedostupný uzel měnil data a zbytek systému by o těchto změnách nevěděl a tak by docházelo k porušení konzistence dat.

- *Uzel byl dočasně nedostupný, ale došlo k obnovení spojení a nyní vidí zbytek systému a může s ním komunikovat.* V takovém případě musíme předpokládat, že během času, kdy nemohl uzel komunikovat s ostatními uzly, mohlo v distribuovaném souborovém systému nastat libovolné množství změn. Znovu dostupný uzel nemusí mít aktuální stav logických hodin a zároveň nemusí

mít synchronní lokální databázi ani frontu databázových požadavků *sync-db*, ze které by mohl stav databáze aktualizovat. Neaktualní stav lokálních logických hodin není potřeba řešit, neboť tyto se aktualizují s příchodem prvního požadavku na synchronizaci. Co ale je nutné řešit, je stav lokální databáze s metadaty. Právě pro tento případ existuje fronta *archiv-db*. Po startu a obnovení konektivity se Sync modul jako první operaci po navázání spojení se svým okolím zeptá všech uzlů systému, jaký mají aktuálně stav logických hodin. Z došlých odpovědí vybere uzel s nejvyšší hodnotou logických hodin pro databázové operace. V případě, že více uzlů odpovědělo stejnou hodnotou, bude vybrán uzel, který odpověděl nejrychleji, neboť se dá předpokládat, že bude dostupný nejvýhodnější datovou cestou z hlediska rychlosti a propustnosti. Po výběru uzlu pro synchronizaci je tento uzel požádán o zaslání požadavků z fronty *archiv-db*. Tyto požadavky jsou rovnou po přijetí umístěny do fronty sync-db, odkud jsou zpracovávány. Díky tomu, že zpracování probíhá sekvenčně, je možné už přijímat žádosti o hlasování o časových značkách a tedy aktivně se účastnit hlasování. K synchronizaci nedojde okamžitě, ale postupně po provedení všech operací a tím i vyprázdnění fronty *sync-db*.

- *Uzel není offline, ale potýká se s výpadky konektivity nebo s lokální zátěží, například v důsledku rebuildu diskového pole, což je běžná situace při rozšiřování kapacity, či výměně disků.* Důsledkem těchto aktivit bude docházet k postupnému zpožďování či nedoručení/nezpracování zpráv od ostatních uzlů. To, že některé zprávy synchronizující požadavky od uzlů nedorazily, nevadí. K opětovné synchronizaci lokálních logických hodin dojde s příchodem dalšího požadavku či vzniku lokálního požadavku na sychronizaci. Ale stejně jako v případě obnovení činnosti nedostupného uzlu, která je popsaná v předchozím bodě 2., je zásadní problém nekonzistentní stav databáze a ztráta některých požadavků na synchronizaci databáze. V případě, že takovýto uzel obdrží zprávu na synchronizaci časových hodin, která je vyšší více než o 1, oproti jeho lokálním logickým hodinám, musí si, stejně jako v případě obnovy systému po výpadku, vyžádat od ostatních uzlů veškeré operace provedené mezi aktuálním lokálním časem T_x a časem obsaženým v příchozí zprávě T_{msg}. Během synchronizace, která s ohledem na množství zpráv, které je nutné synchronizovat, může trvat různě dlouhý čas, může dojít k opakování výpadku. V takovém případě je nutné celou operaci opakovat a to tak dlouho, dokud nedojde k obnovení konzistentního stavu lokální databáze a vyprázdnění front požadavků.

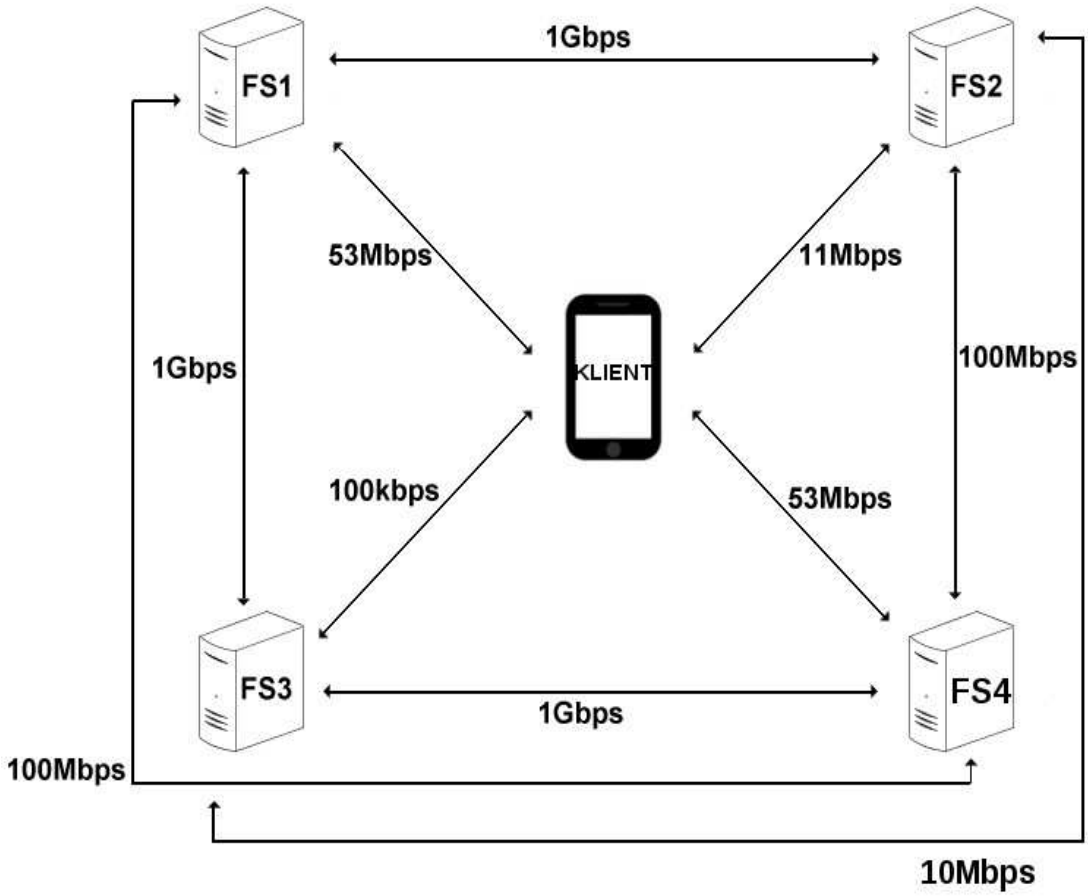
- *Uzel nemá možnost komunikovat s ostatními uzly systému, díky čemuž nedojde nikdy k potvrzení všech synchronizačních zpráv.* Jak již bylo uvedeno, situace kdy jeden nebo více uzlů distribuovaného souborového systému nejsou v daný okamžik schopny komunikovat se zbytkem systému, je poměrně běžná. Tato skutečnost výrazně limituje celkovou propustnost a stabilitu systému, neboť výpadek jediného uzlu by způsobil zastavení veškerých činností v systému. Obdobný problém nastává v situaci, kdy je jeden uzel výrazně přetížený nebo je dostupný pouze prostřednictvím saturované datové linky. Důsledkem těchto negativních vlivů bude zpožděné či úplně zastavené doručování synchronizačních zpráv. Algoritmus hlasování musí nutně s

takovou situací počítat. Základním požadavkem, který je nutné splnit je zachování zvoleného konzistenčního modelu dat a metadat v distribuovaném souborovém systému. Tento požadavek je možné díky transparentnosti přístupu k datům z pohledu uživatele, popsaném v kapitole 5.7, implementaci VFS, které bude popsané v kapitole 7.5, a replikaci metadat popsané v kapitolách 7.6, splnit bez nutnosti získání odpovědí od všech uzlů. Požadavek na potvrzení od všech uzlů je příliš silné omezení. Pro správné fungování hlasování je postačující zajistit, že v rámci KIVFS nemohou vznikat skupiny uzlů, které si samostatně odhlasují stejné operace v rozdílných časech nebo naopak souběžné operace ve stejných časech. Tento požadavek je splněn už v situaci, kdy v rámci hlasování budeme vyžadovat nadpoloviční většinu hlasů, neboť v systému, ať už se sudým či lichým počtem uzlů, nemohou nikdy ve stejný okamžik vzniknout dvě skupiny uzlů, které by obsahovaly nadpoloviční většinu všech uzlů. Pokud uzel, který odpovídá pomaleji obdrží informaci o výsledku hlasování, ve kterém je o několik kroků opožděn, stejně jako v předchozím bodě 2. zažádá o synchronizaci požadavků a tím se vrátí do konzistentního stavu.

7.4.4 Hledání optimálních cest

Vzhledem k tomu, že v rámci implentace úplného uspořádání Lamportových skalárních hodin je již implementován mechanismus na rozesílání zpráv všem ostatním uzlům v systému, je tento mechanismus Sync modulem v KIVFS využit ještě k jedné obdobné činnosti. Jak bude popsáno v kapitole 9, je jedním z cílů této práce a projektu KIVFS zvýšit propustnost distribuovaného souborového systému pomocí změny směrování požadavků podle aktuálního stavu sítě. Tento mechanizmus bude označován jako *dynamické směrování požadavků*. Dynamické směrování požadavků pro svoji práci potřebuje znát kvalitu a propustnost linek mezi jednotlivými uzly. V rámci těchto linek následně hledá nejvýhodnější trasu pro přenos dat z hlediska aktuální dostupné rychlosti a propustnosti linek. Samotné vykonávání dynamického směrování je implementováno v FS modulu a bude popsáno v kapitole 7.7. Sync modul neprovádí přímo směrování jednotlivých požadavků a spojení, ale pro potřeby dynamického směrování připravuje směrovací tabulku. Na základě této směrovací tabulky jsou realizována jednotlivá spojení mezi uzly distribuovaného souborového systému. Zdrojem pro tvorbu směrovací tabulky je matice, která obsahuje informace o všech linkách mezi jednotlivými uzly distribuovaného souborového systému. Z této matice jsou pro všechny dvojice uzlů hledány nejvýhodnější cesty z hlediska přenosové rychlosti, propustnosti a latence linek. Jak je vidět na obrázku 7.5, který představuje příklad zapojení serverů a jednoho klienta pro KIVFS, jsou parametry jednotlivých linek v systému rozdílné. Tento model odpovídá reálnému zapojení, neboť kvalita, parametry a využití jednotlivých linek se mění v čase a jejich chování nejsme v případě geograficky rozděleného systému schopni ovlivnit ani předvídat. Pro model uvedený na obrázku 7.5 a za předpokladu, že budeme pro zjednodušení uvažovat maximální přenosovou rychlost média jako jediný parametr, který použijeme k tvorbě směrovací tabulky, bude směrovací tabulka obsahovat hodnoty z tabulky 7.5.

Z obrázku 7.5 je patrné, že mezi jednotlivými uzly distribuovaného souborového systému existuje větší množtví síťových cest než je v tabulce uvedené. Tím,



Obrázek 7.5: Model zapojení KIVFS

že jsou jednotlivé uzly spojené pomocí počítačové sítě, musíme předpokládát, že každý uzel má ke všem ostatním svoji vlastní nezávislou cestu. Z těchto cest vybereme cestu s nejlepšími parametry, podle definovaných kritérií, které budou uvedeny dále. V tabulce 7.5 je pro zjednodušení použito jako jediné hodnotící kritérium kvalita linky a její maximální možná přenosová rychlost. Z uvedené tabulky je zřejmé, že v systému mohou existovat konkurenční cesty se stejným ohodnocením. Tyto vzájemně rovnocenné cesty jsou v tabulce uvedené všechny. Během testování se ohodnocení pouze pomocí maximální přenosové rychlosti ukázalo jako nedostatečné. Narazilo se na několik podstatných problémů:

- Linky mezi jednotlivými uzly nejsou symetrické, základní důvody jsou dva. Prvním důvodem je vlastnost použitých technologií jako je DSL. Druhým důvodem je nedeterministické zatížení linek mezi jednotlivými uzly.

- Maximální přenosová rychlost nevypovídá nic o reálně dostupné přenosové rychlosti, neboť připojení do LAN může mít uzel pomocí 1Gbps portu, ale přístup do WAN už může být zcela jiný, typicky nižší. Stejně jak se liší rychlosti mezi jednotlivými směrovači na celé cestě mezi každými dvěma uzly.

- Reálně dosažitelná rychlost není konstatní, ale v čase se, vlivem ostatního provozu a zatížení jednotlivých linek, mění. Tyto změny jsou nedeterministické a není možné je přesně předpovídat.

Uzly	**FS1**	**FS2**	**FS3**	**DB**	**KLIENT (CLI)**
FS1	-	FS2	FS3	FS3, DB	CLIENT nebo FS3,DB,CLI
FS2	FS1	-	FS1,FS3	FS1,FS3,DB	FS1,CLI
FS3	FS1	FS1,FS2	-	DB	FS1,CLI nebo DB,CLI
DB	FS3,FS1	FS3,FS1,FS2	FS3	-	CLI nebo FS3,FS1,CLI
KLIENT (CLI)	FS1	FS1,FS2	FS1,FS3 nebo DB,FS3	DB nebo FS1,FS3,DB	-

Tabulka 7.5: Zjednodušené směrovací tabulky pro zapojení KIVFS z obrázku 7.5

- Především u geograficky rozsáhlých systémů je podstatná, kromě přenosové rychlosti a zatížení jednotlivých linek, i hodnota RTT (detailně popsané v kapitole 3.2), neboť z pohledu požadavků v geograficky rozsáhlém systému budou výrazně převažovat požadavky malé, jejichž příkladem jsou synchronizační zprávy, které jsou vysokou hodnotou RTT negativně ovlivněné. Hodnota RTT je pro konkrétní médium konstantní, jak ukazuje tabulka 3.2.

Jelikož stav linek nemůžeme brát jako konstantu nastavenou například v konfiguraci, bylo v rámci Sync modulu implementováno *qualityCheckLink* vlákno, které zajišťuje cyklické testování kvality linek ke všem ostatním uzlům v KIVFS. Toto vlákno se řídí dvěma konfiguračními parametry :

- *qualityCheckLinkFreq*: určuje frekvenci testování. Defaultní hodnota je nastavena na 60s. Minimální hodnota tohoto parametru musí být větší než je dvojnásobek hodnoty parametru *qualityCheckLinkTimeout*, aby nedocházelo k násobným spuštěním

- *qualityCheckLinkTimeout*: určuje jak dlouho bude testovací vlákno čekat na odpověd od testovaného uzlu. Pokud testovací vlákno nedostane odpověď od uzlu včas, bude uzel prohlášen za nedostupný

Detekce kvality linek se startuje ihned po startu Sync modulu. Řídící vlákno *qualityCheckLink*, koordinuje jednotlivé testovací operace a vyhodnocení. Z *qualityCheckLink* je spuštěno další vlákno pro každý uzel KIVFS a na něj jsou odeslány dva KIVFS pakety. Pomocí těchto paketů se zjišťuje přenosová rychlost linky a hodnota RTT mezi dvěma uzly. Jedná se o tyto KIVFS pakety:

- KIVFS_ROUTE_SPEED: odešle se 512kB náhodně generovaných dat. Testovaný uzel odpovídá pouze KIVFS hlavičkou. Důvodem odeslání pouze hlavičky je požadavek na co nejrychlejší odpověď, která minimalizuje odchylky v měření z působení asynchronností linek, jak je vidět ve vzorci pro výpočet 7.1.

- KIVFS_ROUTE_PING: paket tohoto typu obsahuje jen hlavičku, cílem tohoto testu je zjistit parametr RTT jednotlivých linek, tedy jak dlouho trvá přenos signálu prostřednictvím testované linky. Snažíme se minimalizovat přenášená data, aby výsledek měření nebyl ovlivněn přenosovou rychlostí linek mezi testovanými uzly.

Po odeslání KIVFS paketu se čeká po dobu *qualityCheckLinkTimeout* na odpověď. Pokud odpověď dorazí, zaznamená se do matice *qualityCheckMat* čas, který byl potřebný k zaslání dat a přijetí odpovědi. Pokud odpověď nedorazí do času *qualityCheckLinkTimeout*, je do matice zaznamenaná hodnota -1, která značí nedostupnost uzlu. Po dokončení měření kvality linek všech uzlů, které v nejhorším případě skončí v čase *t+qualityCheckLinkTimeout*, se na všechny uzly, které odpověděly do času *qualityCheckLinkTimeout*, pošle žádost o zaslání právě naměřených dat. I tato žádost je časově omezena maximálním časem odpovědi *qualityCheckLinkTimeout*, tak aby nedošlo k zablokování systému při čekání na nedostupný či velmi vzdálený/zpožděný uzel. Po přijetí naměřených dat od ostatních uzlů se testovací vlákna uspí na dobu *qualityCheckLinkFreq*. Hodnota *qualityCheckLinkFreq* je nastavitelná v konfiguraci. Jednotlivá vlákna nejsou po skončení jednoho měření ukončena, aby se eliminovala nutná režie na rušení a znovu vytvoření jednotlivých vláken, zůstávají vlákna uspaná pro další použití, ale zároveň neblokují CPU ostatním procesům daného uzlu aktivním čekáním. Po dokončení všech testů a získání informací od ostatních uzlů sestaví vlákno *qualityCheckLink* matici sousednosti vrcholů. Hodnoty v matici udávají ohodnocení linky mezi dvěma uzly. Hodnota ohodnocení se počítá z naměřených hodnot pomocí vzorce 7.1.

$$M = \frac{cas\_prazdne\_zpravy * p}{cas\_zpravy\_s\_daty} * c, kde\ p = \frac{velikost\_dat}{velikost\_hlavicky} \tag{7.1}$$

Každý uzel naplní jeden řádek matice z vlastních naměřených hodnot. Ostatní řádky matice jsou doplněny hodnotami od ostatních uzlů, získaných pomocí dotazu po skončení měření. Z hodnot matice sousednoti vrcholů jsme schopni zkonstruovat neorientovaný ohodnocený graf, kde uzly grafu jsou uzly distribuovaného souborového systému, hrany jsou reprezentovány linkami mezi uzly a ohodnocení odpovídá hodnotám vypočítaných ze vzorce 7.1. Pro systém zobrazený na obrázku 7.5 je matice sousednosti vrcholů znázorněna v tabulce 7.6.

Uzly	**FS1**	**FS2**	**FS3**	**FS4**	**CLIENT**
FS1	0	1	1	1	1
FS2	1	0	1	1	1
FS3	1	1	0	1	1
FS4	1	1	1	0	1
CLIENT	1	1	1	1	0

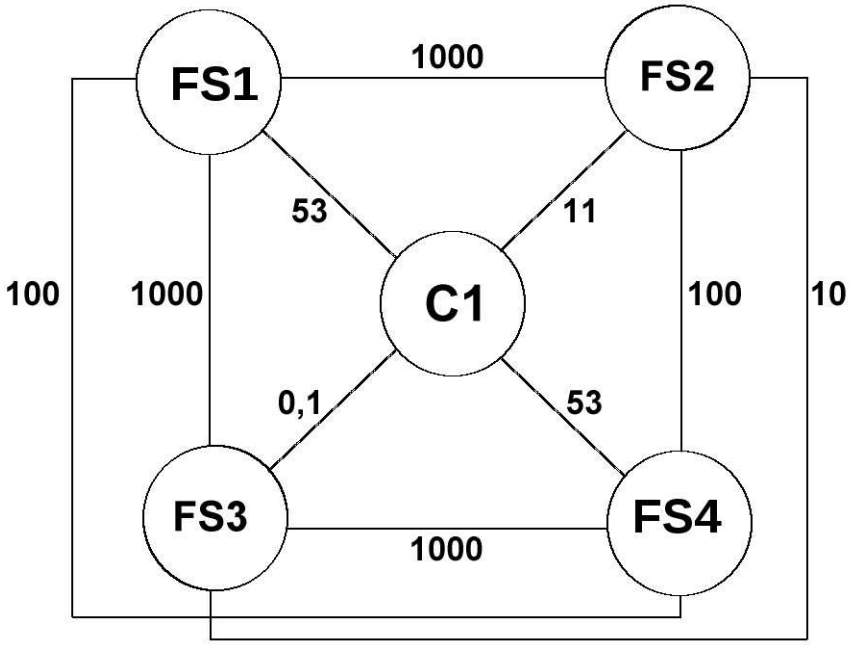
Tabulka 7.6: Příklad matice sousednosti pro zapojení KIVFS z obrázku 7.5

Na základě matice sousednosti lze vytvořit neorientovaný ohodnocený graf, který použijeme jako základní prvek pro dynamické směrování požadavků, neboť algoritmy používané v dynamických směrovacích protokolech vycházejí z teorie

grafů a na stejném teoretickém základě je postaveno i dynamické směrování v KIVFS. Cílem dynamického směrování požadavků v distribuovaném souborovém systému je nalézt takovou cestu mezi uzly distribuovaného systému, kterou může téct maximální datový tok. Tedy cestu s nejnižší latencí a maximální propustností. Zapojení distribuovaného souborového systému, které je znázorněno na obrázku 7.5 převedeme na neorientovaný ohodnocený graf. Při převodu budeme, v rámci zjednodušení, vycházet z předpokladu, že linky mezi uzly jsou symetrické, což je mezi servery obvyklý stav. Pro tvorbu neorientovaného ohodnoceného grafu definujeme trojici udajů:

- Uzly: budou reprezentovány uzly distribuovaného souborového systému
- Hrany: budou reprezentovat přímé virtuální propoje mezi jednotlivými uzly
- Ohodnocení: bude hodnota z výsledku vzorce 7.1 z naměřených hodnot pro latenci a maximální přenosovou rychlost dostupnou mezi dvěma uzly

Výsledný neorientovaný ohodnocený graf pro distribuovaný souborový systém z obrázku 7.5 je znázorněn na obrázku 7.6.



Obrázek 7.6: Neorientovaný ohodnocený graf

Z výsledků hledání minimálních cest je vytvořena směrovací tabulka, jak bude popsáno v kapitole 9.2. Tuto směrovací tabulku využívá FS modul, popsaný v kapitole 7.7, pro realizaci dvou typů požadavků:

- Replikace dat
- Přenos dat pro klienta

Problém hledání nejvýhodnějšího spojení z hlediska maximalizace přenosových rychlostí a minimalizace RTT lze transformovat na problém hledání minimální cesty nebo maximálního toku v síti, což jsou vzájemně převoditelné úlohy

a možnosti jejich řešení jsou známé z teorie grafů a diskrétní optimalizace. Známé algoritmy na řešení tohoto problému jsou :

- Floyd-Warshall algoritmus [87]
- Bellman-Ford algoritmus [87]
- Demetrescu-Italiano algoritmus [88]
- Dijkstrův algoritmus [89]

Algoritmus, který hledáme musí být rychlý a nenáročný na výpočetní zdroje, aby se mohl použít i na klientech, kteří jej také mohou využít k tvorbě směrovací tabulky požadavků. Z výše uvedených algoritmů byl v KIVFS implementován Dijkstrův algoritmus, díky své procesorové a paměťové nenáročnosti.

Pomocí převodu distribuovaného souborového systému na neorientovaný ohodnocený graf vytvoříme směrovací tabulku datových požadavků *routeTable*, která je implementována jako vektor spojových seznamů. Indexy vektoru tvoří jednotlivé uzly, hodnotou vektoru je jednosměrně zřetězený seznam, který určuje jakou cestou mají být data poslána, aby se minimalizoval čas nutný k přenosu. Ukázka definice vektoru směrovací tabulky je znázorněna na výpisu 7.2.

Výpis 7.2: Implementace směrovací tabulky KIVFS

```
typedef typeNode {
        *char name,
        *typeNode nextNode
} struct Node;
Node[countOfNodes] routeTable;
```

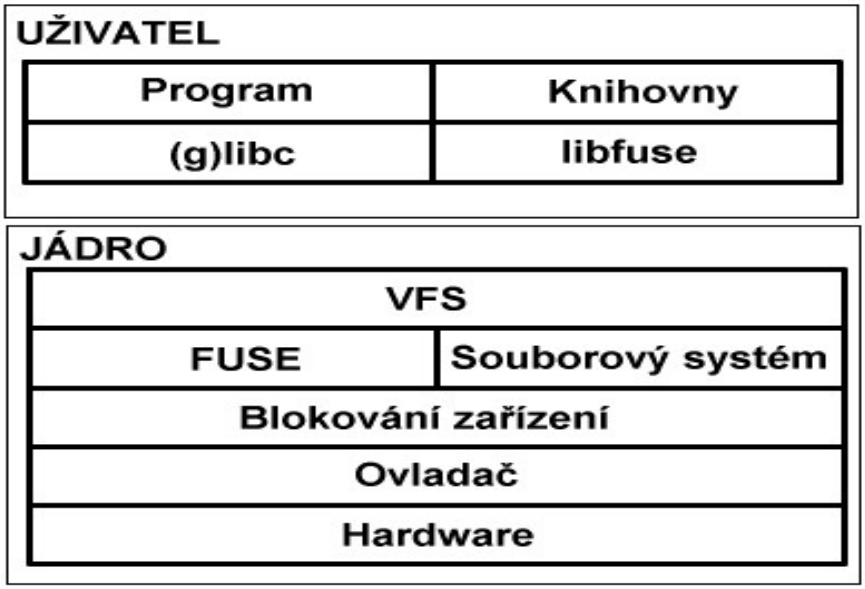
Směrovací tabulka 7.5 v KIVFS umožní jednoduše využít nejvýhodnější spojení, z hlediska přenosových charakteristik datové cesty a v KIVFS je použitá pro realizaci dynamického směrování datových požadavků, který je popsán v kapitole 9.

7.5 VFS modul

Virtuální souborový systém, označovaný jako VFS, je běžnou součástí jádra většiny operačních systémů. Tato vrstva jádra operačního systému definuje jednotné API pro zajištění transparentnímu přístupu k jednotlivým konkrétním souborovým systémům, jak je znázorněno na obrázku 7.7 pro operační systém GNU/Linux.

Pomocí VFS modulu je v distribuovaném souborovém systému KIVFS realizována abstrakce přístupu a zpracování dvou typů dat souborového systému:

- *Data*, což je obsah jednotlivých souborů jsou poskytována *FS modulem*, který je popsán v kapitole 7.7
- *Metadata*, což jsou doplňkové informace k souborům a adresářům, jako je název, vlastník, velikost, čas vzniku atd. Tato data jsou poskytována DB modulem, který bude popsán v kapitole 7.6.



Obrázek 7.7: VFS v GNU/Linux

VFS modul kontroluje požadavky a podle obsahu jej předá na FS nebo DB modul. Důvodem tohoto rozdělení je skutečnost, že velká většina požadavků je směrována na DB modul, který s využitím cache v operační paměti může reagovat výrazně rychleji, než FS modul pracující s většími daty na pevných discích. Efektivní využití cache je možné proto, že metadata i když je jich velké množství z pohledu počtu záznamů, nejsou v porovnání se samotnými daty, přiliš rozměrná. Naproti tomu FS modul je výrazně pomalejší, neboť veškerá jeho data jsou uložena na pevných discích a cache se s ohledem na možný rozsah dat použije jen omezeně. Proto se operace FS modulu volají jen v případě, když je požadován obsah konkrétního souboru. Toto rozdělení je realizováno na základě konkrétních volání KIVFS protokolu, kde pouze tři, které jsou uvedeny v tabulce 7.7, jsou směrované na FS modul.

Příkaz	**Význam**
KIVFS_FS_READ	Čtení souboru
KIVFS_FS_DELETE	Smazání souboru
KIVFS_FS_WRITE	Uložení obsahu souboru

Tabulka 7.7: Volání KIVFS protokolu, která jsou obsloužena FS modulem

7.6 DB modul

DB modul slouží jako úložiště metadat. Je tvořen dvěma samostatnými programovými částmi:

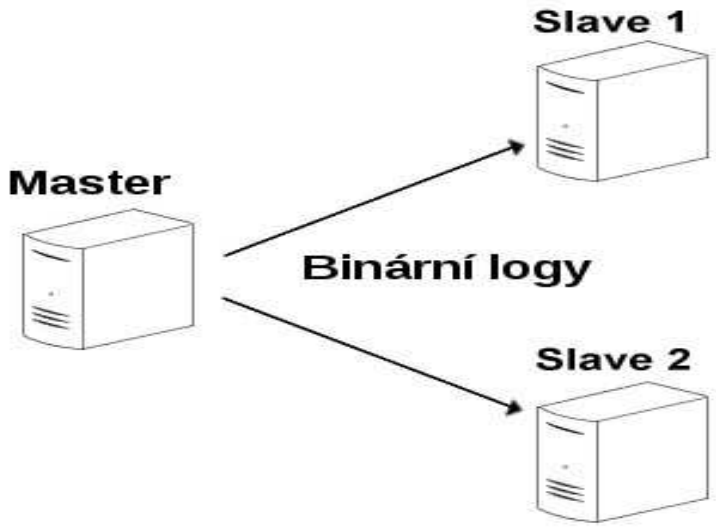
- Wrapper nad konkrétním databázovým systémem, který transparentně překrývá konkrétní databázový program a jeho odlišnosti v implementace jazyka SQL. Tento program komunikuje s VFS modulem pomocí KIVFS

protokolu a zároveň pomocí UNIX socketů nebo TCP/IP protokolu s databázovým systémem. V rámci testování výkonnosti a škálovatelnosti byl implementován wrapper pro následující databázové systémy:

  - MySQL [90], které je zvoleno jako výchozí varianta databázového řešení v KIVFS, díky své jednoduchosti a úspornosti při jednoduchých operacích, které jsou v KIVFS požadovány.
  - PostgreSQL [91]
  - Microsoft SQL Server [92]
  - Oracle DB [93]

- Databázový systém, tedy systém pomocí kterého je realizovaná samotná správa a organizace dat v databázi.

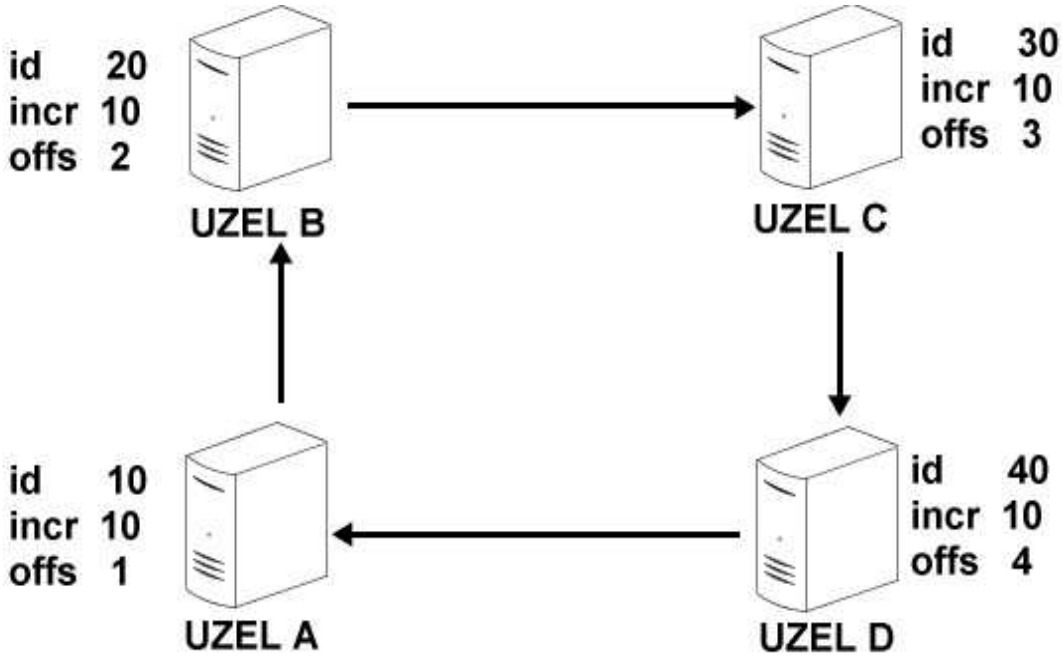
Aktuálně používaný databázový systém MySQL sice poskytuje základní možnost práce v clusteru či replikaci dat, stejně jako ostatní podporované systémy, ale tato vlastnost není v KIVFS záměrně využívaná. Při replikaci dat, tak jak je v MySQL či PostgreSQL podporována, není zajištěna synchronní replikace napříč všemi uzly databázového systému v reálném čase. V MySQL je replikace realizována tak, že jsou uzly rozděleny na jeden master a N slave uzlů. Master uzel vytváří binární log operací, který je postupně přehráván na slave uzlech, tak jak ukazuje obrázek 7.8. Veškeré operace zápisu jsou prováděny pouze na master uzlu a slave uzly slouží jako read-only repliky.



Obrázek 7.8: Master-Slave replikace v MySQL

Jak je z obrázku 7.8 patrné, na slave uzlech může nastávat zpoždění při přehrávání replikačního logu z master uzlu. Toto zpoždění by se v systému projevilo jako nekonzistence metadat v distribuovaném souborovém systému. Popsaná konfigurace je navíc v módu master - slave, kde slave uzly jsou pouhou read-only kopií. Každý jednotlivý uzel KIVFS disponuje svojí vlastní plnohodnotnou kopií databáze, se kterou potřebuje pracovat. Je tedy nutné zapojení master - master, což MySQL podporuje, ale s dalšími omezeními. Tím základním je opět asynchronní zpracování dat, tedy operace se provede na jednom uzlu a replikuje se na další,

stejně jako v případě master - slave replikace. V modelu master - master už je ale možné, aby další uzly vytvářely své lokální operace a ty se přenášely na ostatní uzly. Jednotlivé změny se systémem šíří postupně, jak ukazuje obrázek 7.9.



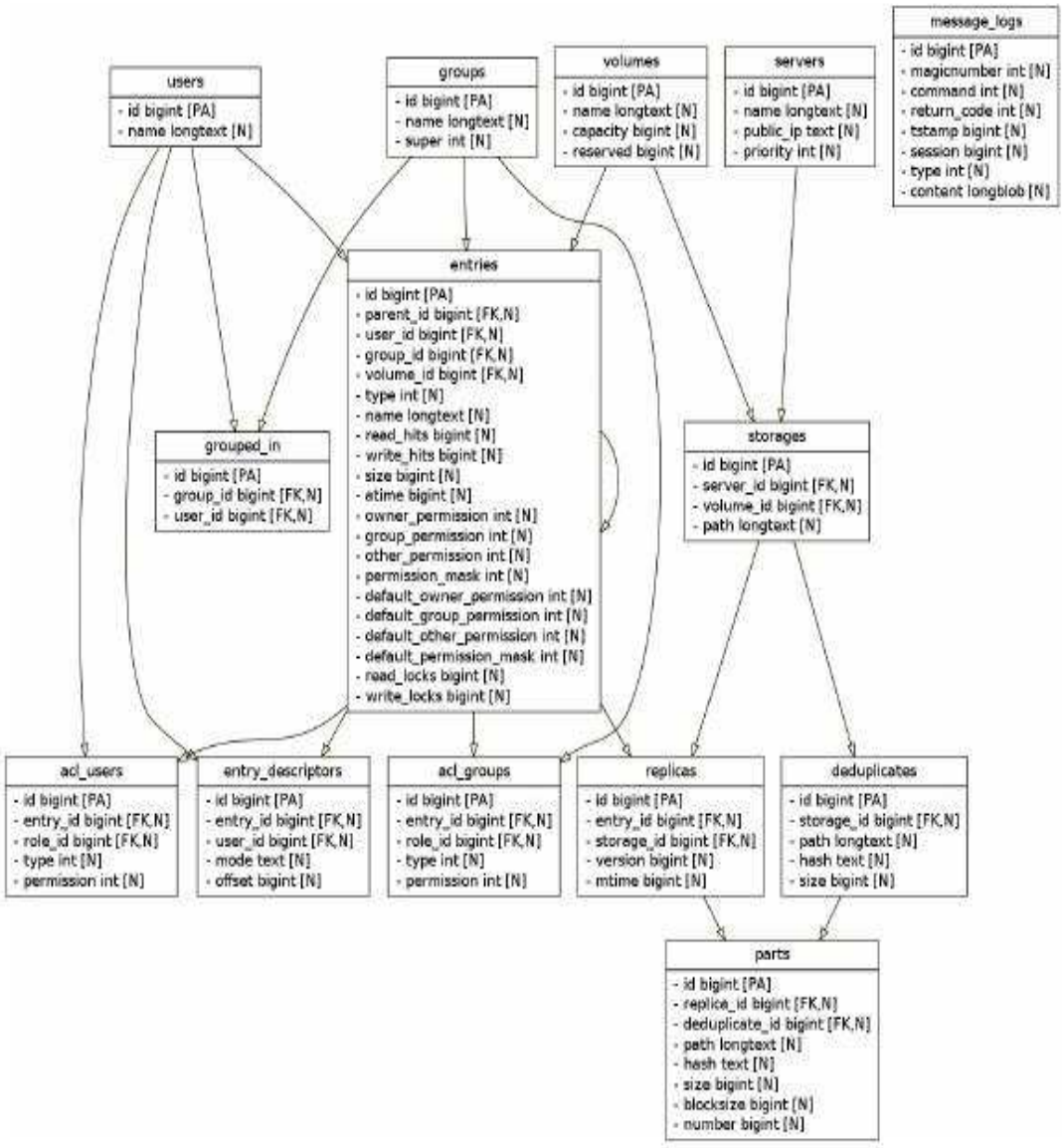
Obrázek 7.9: Multi master - master replikace v MySQL

Ani v tomto zapojení však není zajištěna možnost synchronních operací na všech uzlech. Navíc tím jak jednotlivé uzly pracují samostatně a ostatním jen předávají provedené změny, vzniká problém s unikátností primárních klíčů v datových tabulkách, neboť v případě, že je auto-increment primárního klíče nastaven na 1 a na dvou uzlech dojde zároveň či téměř zároveň k tvorbě nového záznamu, dostanou oba záznamy stejné id na více uzlech a v daný okamžik selže replikace, neboť při přehrátí transakčních logů vznikne požadavek na tvorbu dvou záznamů se stejným primárním klíčem. Tento konkrétní problém se částečně ošetřuje změnou výchozí hodnoty, o kterou je proveden auto-increment. Tato hodnota musí být nastavena různě na různých uzlech. Problém se úplně nevyřeší, ale výrazně se sníží pravděpodobnost jeho vzniku. Což ale není vhodné řešení pro distribuovaný souborový systém, kde pro zachování funkčnosti systému a konzistence dat je nutné zajistit, aby tento problém nemohl nastat. Jelikož takovýchto omezení je v existujících řešeních více a není tedy zcela spolehlivě zajištěno, že operace budou prováděny synchronně a v přesném pořadí na celém KIVFS, je v KIVFS použit vlastní synchronizační mechanismus, který využívá Sync modul popsán v kapitole 7.4.

S ohledem na možnost přenosu na jiné databáze, je ERA model řešen minimalisticky. V návrhu databáze, jak je vidět na obrázku 7.10, jsou uchovávány jen nejnutnější informace o souborech a jejich atributech. Záměrně nejsou používány žádné složitější konstrukce, jako vložení procedury či vlastní datové typy, které by jednak zpomalovaly databázi, ale také komplikovaly případný přechod na jiný databázový systém. Základní význam jednotlivých tabulek je uveden v tabulce 7.8

Název tabulky	**Popis**
USERS	Uživatelé
GROUPS	Skupiny uživatelů
GROUPED_IN	Vazba M:N mezi *USERS* a *GROUPS*
VOLUMES	Virtuální svazky, jakožto největší mapovatelná datová jednotka
SERVERS	Souborové servery definované v jedné skupině KIVFS
MESSAGE_LOGS	Trvalý log operací pro možnosti obnovy uzlu či přidání a dosynchronizace nového uzlu, které je popsané v kapitole 7.4.3
ENTRIES	Definice adresářů a souborů
STORAGES	Informace o datových úložištích na jednotlivých souborových serverech
REPLICAS	Informace o uložení jednotlivých svazků na souborových serverech. Definuje vztah M:N mezi VOLUMES a SERVERS
ACL_USERS	ACL práva uživatelů na jednotlivé adresáře a soubory
ACL_GROUPS	ACL práva skupin na jednotlivé adresáře a soubory
DEDUPLICATES	Informace o slučování binárních bloků - deduplikace dat, více v 7.7
ENTRY_DESCRIPTION	Informace o otevřených souborech
PARTS	Mapování jednotlivých binárních částí souborů na metadata souboru, více v 7.7

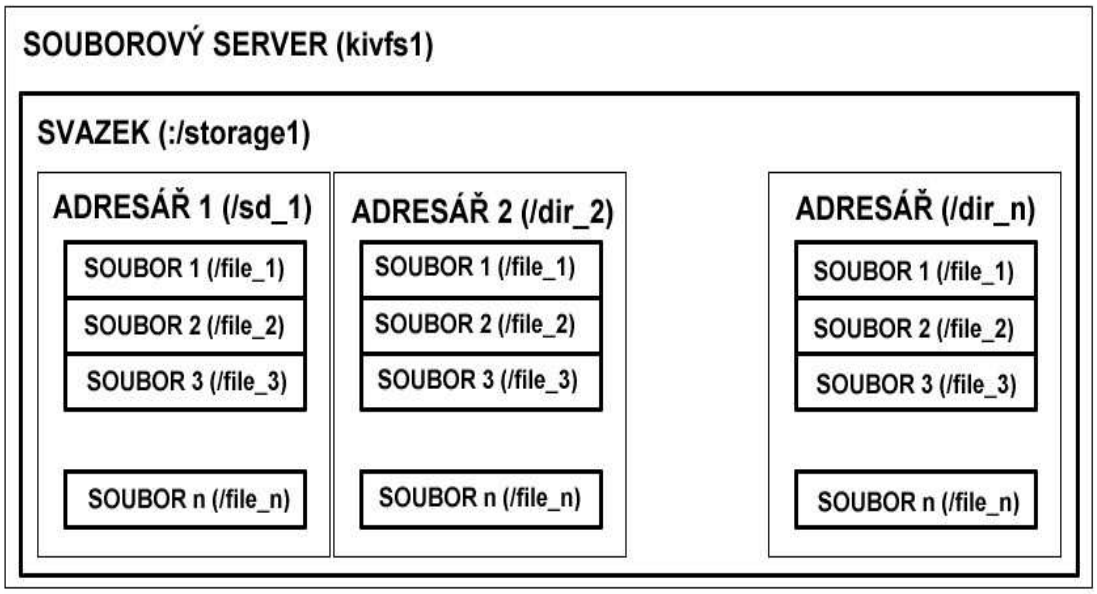
Tabulka 7.8: Popis jednotlivých tabulek v ERA modelu KIVFS z obrázku 7.10



Obrázek 7.10: ERA model KIVFS, zdroj [68]

Jak ukazuje ERA model na obrázku 7.10 a tabulka 7.8 veškeré informace o celém souborovém systému, tedy existence souborů, adresářů, jejich vzájemné mapování, mapování adresářů na svazky a svazky na souborové servery, stejně jako popis replikací a zámky nad soubory jsou obsaženy v databázi. Tím je v databázi definována logická struktura souborového systému, jak je znázorněno na obrázku 7.11

Těchto informací je sice hodně, ale v porovnání s fyzickým obsahem souborů se jedná o data o několik řádů menší. To je výhodné, neboť je reálné mít pomocí cache většinu těchto dat obsaženou v operační paměti jednotlivých uzlů a tím výrazně urychlit veškeré operace s metadaty.



Obrázek 7.11: Logická struktura distribuovaného souborového systému

7.7 FS modul

FS modul je posledním článkem celého řetězce modulů v KIVFS. Tento modul má na starosti fyzické uložení obsahu souboru, neboť jak bylo uvedeno v předchozích kapitolách, kompletní metadata celého souborového systému jsou z důvodů zvýšení propustnosti systému uložena v databázi. Z požadavků klienta se FS modulu dostanou jen tři operace, které jsou uvedeny v tabulce 7.7. Obsluha požadavků klienta je jen malá část operací, které FS modul zajišťuje. Základní činnosti FS modulu jsou uvedeny v následujícím přehledu:

- Práce s obsahem souboru dle požadavků od klienta uvedených v tabulce 7.7.
- Dělení souborů na části a jejich samostatná správa, nutná pro efektivní replikaci a deduplikaci.
- Zajištění deduplikace dat souborového systému na úrovni binárních souborových bloků.
- Zabezpečení uložených dat pomocí šifrování.
- Zajištění replikace dat mezi jednotlivými uzly distribuovaného souborového systému.
- Dynamické směrování datových požadavků.
- Multi-Master RW On-line replikace

FS modul se stará o manipulaci s daty a spolu s Sync module je pro tuto práci klíčový.

7.7.1 Obsluha požadavků klienta

Pokud požadavek od klienta dorazí až k FS modulu, jedná se o jeden z požadavků uvedených v tabulce 7.7. Jednotlivé požadavky budou obslouženy podle následujících scénářů:

- KIVFS_FS_READ: Tato operace nastane, pokud klient požaduje obsah datového souboru. V takovém případě se FS modul dotáže DB modulu na umístění uložení všech fragmentů požadovaného souboru (více o fragmentech v kapitole 7.7.2. Poté je vybrán náhodný volný port. FS modul spustí nové vlákno, které bude naslouchat na zvoleném portu. Informace o zvoleném portu a IP adresa souborového serveru, který tento požadavek vyřizuje budou předány klientovi. Klient se pomocí získaných údajů připojí k FS modulu, provede se přenos požadovaného obsahu. Komunikace klienta a serveru může být šifrovaná. Šifrování zvyšuje zabezpečení datového přenosu, ale snižuje přenosovou rychlost a zvyšuje nároky na procesor. Proto je možné šifrování v některých speciálních případech, jako jsou uzavřené systémy, či systémy dostupné jen prostřednictvím vyhrazené VPN, vypnout pomocí konfigurační direktivy *crypto=[0/1]*. Po dokončení přenosu je vlákno deaktivováno a port uvolněn. Pro další přenos souboru je vybrán další náhodný port. Jednotlivá vlákna se kompletně neruší, ale jsou udržována pro další použití, čímž šetříme režií nutnou k tvorbě a rušení vláken. Volba nového náhodného portu pro každý jednotlivý přenos, podobně jako šifrování komunikace, zvyšuje zabezpečení komunikace celého systému.

- KIVFS_FS_DELETE: Po přijetí požadavku na smazání souboru je nejprve dotázán DB modul na seznam všech fragmentů souborů a ty jsou postupně odstraněny. Teprve poté co je vrácena informace o úspěšném smazání všech fragmentů souboru, je možné provést odstranění i metadat souboru z interní databáze. V rámci mazání je potřeba ošetřit ještě stav, kdy je díky defragmentaci (blíže popsané v kapitole 7.7.2) nutné některé fragmenty neodstraňovat neboť jsou součástí více souborů. Tuto situaci řeší již DB modul a při předání seznamu všech fragmentů na smazání automaticky vynechá ty, které jsou v čase mazání požadavku přidruženy k jinému než právě mazanému souboru.

- KIVFS_FS_WRITE: V případě zápisu obsahu souboru je postup obdobný jako při čtení dat. Je vybrán náhodný port, který je doručen klientovi, probuzené vlákno začne naslouchat na zvoleném portu a čeká na klienta. Po připojení klienta jsou postupně přenášena data. Název souboru, stejně jako cesta v rámci souborového serveru, kam jsou data ukládána, je předána z DB modulu. FS modul příchozí data ukládá po pevně daných fragmentech (více v kapitole 7.7.2) a po uložení každého fragmentu se tato informace o identifikaci a stavu fragmentu předá zpět DB modulu. Ihned po dokončení uložení jednoho celého fragmentu je spuštěna replikace dat (více v kapitole 7.7.5). Pro ukládání dat mohou nastat dvě situace:

  - Jedná se o nový soubor, v takovém případě jsou postupně ukládána a předávána veškerá data.

– Jedná se o aktualizaci stávajícího souboru. V tomto případě nedojde k přenosu celého souboru, ale v rámci úspor datových přenosů probíhá komunikace s klientem a konkrétně klientskou cache, pomocí níž se FS modul informuje o verzích jednotlivých fragmentů souboru v klientské cache a k fyzické změně, a tedy i datovému přenosu, dojde jen u těch fragmentů, které se liší. Tímto postupem budeme přenášet jen opravdu změněná data, což nám umožní výrazným způsobem snížit datové přenosy.

7.7.2 Fragmentace datových souborů

Souborový systém by měl být schopen pojmout soubor téměř libovolné délky. U klasických souborových systémů existují na délku souboru různá implementační omezení, ale jedno omezení mají jednotlivé systémy společné a tím je maximální velikost disku nebo diskového oddílu, na který data ukládáme. Toto omezení nemusí nutně platit pro distribuovaný souborový systém, neboť můžeme využít více diskových oddílů na více souborových serverech a jeden soubor rozložit na více menších fragmentů. Tento postup nazýváme fragmentací souboru. Fragmentace může nastávat až v případě, že na disku není dostatek volného místa. V KIVFS se fragmentace využívá ještě k dalšímu účelu a to je optimalizace výkonu. Fragmentací v KIVFS vznikají datové bloky souborů přesné velikosti, což nám přináší tři zásadní výhody:

- Na klientské straně můžeme přednačítat samostatně data do cache po jednotlivých fragmentech a to z různých souborových serverů, což z pohledu klienta povede k zrychlení celého systému.
- Díky fragmentaci je možné replikovat data po částech (jak bude detailně popsáno v kapitole 9.3), což povede k výraznému urychlení konvengence systému do konzistentního stavu a tedy dosažení plné replikace.
- Proces fragmentace vytváří stejně velké datové bloky, které je možné binárně porovnat a na základě binární shody provést deduplikaci dat, která je popsána v následující kapitole 7.7.3 a která vede k úspoře místa na datových úložištích.

V KIVFS probíhá fragmentace v průběhu nahrávání obsahu souboru bez ohledu na celkovou velikost dat. Velikost fragmentu je možné nastavit pomocí konfigurační direktivy *fragment=SIZE*, kde *SIZE* je velikost fragmentu v MB.

Pro každý fragment se ihned po dokončení nahrávání spočítá SHA256 otisk. Tento výpočet je prováděn průběžně jak data přicházejí, a tak není po dokončení nahrávání dat nutné čekat na provedení výpočtu. Tento otisk je unikátní pro každý fragment a jeho hodnota je uložena v databázi spolu s dalšími metadaty a slouží jako základní podklad pro deduplikaci fragmentů popsanou v následující kapitole 7.7.3.

7.7.3 Deduplikace souborových fragmentů

Tím, že distribuovaný souborový systém může používat velké množství uživatelů, kteří jsou typicky sdruženi v jedné organizační jednotce, jako je univerzita nebo

firma, kde pracují s podobným softwarem nebo na společných projektech, je velmi pravděpodobné, že se v rámci distribuovaného souborového systému budou jednotlivé soubory nebo alespoň jejich části opakovat. Nejběžnějšími příklady jsou například:

- Výchozí soubory v domovských adresářích, které vznikají při používání určitého softwaru, například .bashrc nebo .profile.
- Násobná kopie dokumentů nebo dat, se kterými pracuje společně jeden tým a jejichž kopie mají jednotliví uživatelé ve svých domovských složkách. Například lokální kopie gitu[94] nebo subversion[95] repository.
- Sdílené dokumenty jako jsou multimedia.

To jsou vše příklady dat, které ve zcela identických kopiích zabírají místo na disku nejen v produkčním systému, ale i v zálohách. Možností řešení tohoto problému je v distribuovaném souborovém systému proces deduplikace dat. Deduplikační proces je možné provést díky transparentnosti přístupu k datům z pohledu uživatele. Deduplikace je prováděma tak, že v rámci souborového systému najdeme shodné datové bloky a ty nahradíme odkazem na jeden společný. V Unixových systémech se tento postup může v rámci jednoho svazku souborového systému provádět pomocí *hard linků*. V distribuovaných souborových systémech, kde mohou být data uložena na více svazcích a na více uzlech, není systém hard linků dostatečným ani proveditelným řešením. V KIVFS je deduplikace implementována jako externě vyvolaná operace, kterou je možné spouštet pomocí plánovače. Deduplikace v KIVFS má dvě části:

- Hledání shodných částí dat. Tato operace se provádí nad DB modulem. Základní jednotka deduplikace je jeden fragment a cílem operace je hledat binárně shodné fragmenty a v případě, že jsou takové fragmenty nalezeny, tyto nahradit odkazem na jedinou kopii. Seznam všech fragmentů je obsažen v databázi, stejně jako jejich SHA256 otisk, který je unikátní pro obsah datového fragmentu. Deduplikace hledá shodné fragmenty pomocího tohoto otisku a nahrazuje je odkazem na nejstarší nalezený fragment. Nejstarší existující fragment byl jako cíl odkazů zvolen z toho důvodů, že je u něj nejmenší pravděpodobnost změny a tím se snižuje pravděpodobnost nutnosti upravovat odkazy duplicitních fragmentů na jiný společný fragment.
- Vymazání obsahu framentu, probíhá na FS modulu, kam je postupně, po nalezení všech duplicit pro jeden fragment, zaslán příkaz k vymazání framentu.

Jelikož distribuovaný souborový systém typicky obsahuje velké množství dat, běžně v jednotkách tera bytů, je operace deduplikace časově i výkonově náročná. V KIVFS se tento problém řeší postupnou deduplikací. Při každém dalším spuštění deduplikace vyhledává shodné fragmenty jen k datům, která byla změněna od posledního běhu deduplikace. Tato informace je pro jednotlivé soubory obsažena v metadatech uložených v databázi. Tím se celá operace, v závislosti na počtu změn v systému, může výrazně urychlit.

Další podstatný moment při použití deduplikace je aktualizace dat. Pokud uživatel maže nebo modifikuje soubor, který byl deduplikován, je nutné před

změnou deduplikovaného fragmentu zkontrolovat, zda se jedná pouze o odkaz na fragment nebo samotný fragment. Pokud fragment, se kterým pracujeme je deduplikovaný a je odkazován na společný zdroj, můžeme tento odkaz bez dalších problémů zrušit nebo nahradit novým fragmentem s novým obsahem. Pokud se ale jedná o fragment zapojený do deduplikace, který slouží jako cíl pro odkazy jiných deduplikovaných fragmentů, je u něj k dispozici hodnota, která udává počet odkazujících se fragmentů. Pokud je tato hodnota rovna nule, můžeme fragment změnit. Pokud není a tedy se na fragment odkazuje jiný fragment, je vybrán první fragment, který se na zpracovávaný fragment odkazuje a ten je z odkazu změněn na fyzický fragment. Pak můžeme aktuálně zpracovávaný fragment modifikovat, neboť už není součástí deduplikace.

Deduplikace, kromě již zmíněné úspory místa na datovém úložišti, má pozitivní vliv i na propustnost a výkonnost distribuovaného souborového systému. Pokud vezmeme jako příklad již zmiňovaný soubor .bashrc, je tento soubor načítán při každém přihlášení všech uživatelů na unixových systémech, což je poměrně častá operace. Při 100.000 uživatelích je soubor 100.000 krát umístěn v různých částech datového úložiště a s přístupem každého jednotlivého uživatele musí být nalezen na datovém úložisti a doručen ke klientovi a vícekrát se po dobu přihlášení nepoužije. Tento soubor nebude opakovaně čten dalšími uživateli z cache, neboť se z pohledu souborového systému jedná o 100.000 různých souborů, byť se stejným obsahem. Naproti tomu pokud tento soubor bude deduplikován, na datovém úložišti se bude nacházet jen jednou. Přihlášení prvního uživatele jej načte do cache disku nebo souborového serveru a každý další přístup už je odbaven z cache a tedy násobně rychleji. Stejně tak tomu je v případě terminálových serverů, kde se na jeden server, který má jako domovský adreář připojen KIVFS, hlásí více uživatelů. Pak je tento soubor odbaven poprvé ze serveru, ale podruhé už z cache klienta[75].

7.7.4 Šifrování fragmentů souborů

Kromě bezpečnosti dat z pohledu síťového provozu, a tedy zajištění šifrování komunikace a omezení práv jednotlivých uživatelů pomocí ACL, jak již bylo popsáno v předchozích kapitolách, je vhodné zabezpečit data i na souborovém serveru. Zabezpečení dat na souborových serverech proti lokálnímu zneužití je v KIVFS zajištěno šifrováním fragmentů souborů při ukládání ve FS modulu. Šifrování je realizováno šifrou AES256[41]. Tato šifrovací metoda byla zvolena na základě experimentů v práci [71], jejichž výsledky jsou shrnuty v tabulce 7.2.

Jelikož zvolená šifrovací metoda umožňuje průběžné zpracování dat, jsou data šifrována tak, jak přicházejí od klienta a ukládána přímo v šifrované formě. Eliminuje se tím jednak nutné zdržení, způsobené následným šifrováním dat po uložení, druhak možnost zneužití dočasných souborů před zašifrováním. Šifrování bohužel vnáší do celého procesu zpoždění dané výpočtem šifrovaných dat. Proto byla do konfigurace systému přidána možnost šifrování ukládaných dat vypnout. Jedná se o parametr *Crypt=0—1* v sekci *FS* konfiguračního souboru. Ve výchozím nastavení je tato možnost vypnuta, neboť se předpokládá provoz serveru v zabezpečeném prostředí a je žádoucí výchozí konfiguraci optimalizat pro vyšší přenosové rychlosti.

7.7.5 Replikace dat

Kromě manipulace s lokálními daty se FS modul stará i o zajištění replikací dat na ostatní FS servery. Replikace, jak bylo uvedeno v kapitole 5.3, je jednou z nejpřínosnějších funkcí distribuovaných souborových systémů zajišťující vyšší zabezpečení dat a umožňuje zvýšit propustnost systému. Replikace je v rámci FS modulu pouze prováděna, její konfigurace je uložena v databázi, ke které se přistupuje prostřednictvím DB modulu. Replikace je v KIVFS nastavována na úrovni svazků, což je nejmenší replikovatelná jednotka a je realizována jako *Multi-Master Online RW replikace* jak bude popsáno v samostatné kapitole 9.3.

7.7.6 Dynamické směrování datových požadavků

Poslední funkcí FS modulu je zajištění dynamického směrování datových požadavků. Tato funkce je v KIVFS implementována na základě spolupráce více modulů. V FS modulu je implementováno samotné směrovaných datových požadavků. Toto směrování je implementováno na základě směrovací tabulky popsané v kapitole 9.2. Podrobně bude dynamické směrování datových požadavků popsáno v kapitole 9.

7.8 Klienti

V současné době je pro KIVFS implementováno více klientů pro různé platformy. Aktuální seznam dostupných implementací klientů je uveden v tabulce 7.9.

Platforma	**Popis**
Windows Mobile	Nativní aplikace v jazyce C#
Android	Aplikace pro mobilní zařízení v jazyce Java
Microsoft Windows	Modul pro souborový manažer Total Commander v jazyce C++
WWW	Klient pro práci s KIVFS prostřednictvím webového prohlížeče implemetovaný v Java, JavaScript, Ajax
GNU/Linux Fuse	Klient implementovaný prostřednictvím technologie Fuse a podporuje základní offline operace
GNU/Linux	Nativní klient implementovaný jako modul jádra operačního systému
GNU/Linux	Testovací klient napsaný v jazyce C. Umožňuje dávkové zpracování příkazů a slouží pro testování jednotlivých částí KIVFS

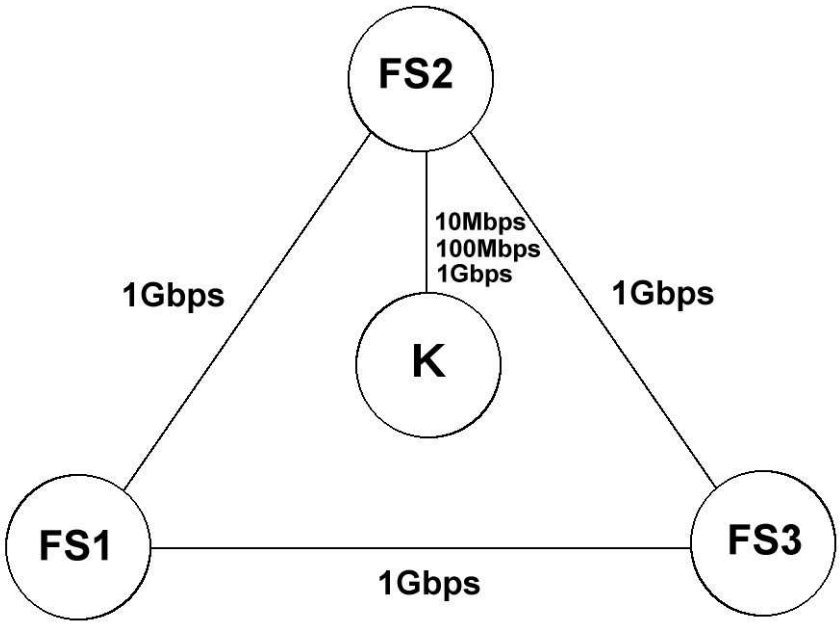
Tabulka 7.9: Dostupní KIVFS klienti v roce 2016

Všechny implementace klienta jsou schopny základní funkčnosti, tedy ověření uživatele, přenosu dat a vytváření dat. Ovládání jednotlivých klientů se liší na základě zvyklosti dané platformy. Společnou mají základní konfiguraci, což zjednodušuje přechod mezi jednotlivými platformami. Pro jednoduchost byl zvolen formát INI souborů[96]. Většina klientů komunikuje pomocí *KIVFS protokolu*. Při implementaci webového klienta se ukázalo, že jednodušší cesta implementace

je využití aplikačního proxy serveru. Tento proxy server implemetuje převodník mezi *KIVFS protokolem* na jedné a *HTTP protokolem* na druhé straně. Vytvořený proxy server byl následně použit i pro implementaci klienta pro platformu Android, která jako základní komunikační prostředek využívá *REST API*[83] přenášené pomocí *HTTP* protokolu. V současném roce 2017 probíhá implementace nativního klienta pro Microsoft Windows a nativního klienta pro mobilní zařízení s operačním systémem iOS.

7.9 Výkonnostní testy

Pro ověření správnosti návrhu a implementace byly provedeny dvě sady testů, které měly za cíl ověřit stabilitu a propustnost distribuovaného souborového systému KIVFS a zároveň porovnat jeho výkonnostní parametry s již existujícím řešením. Jako referenční příklad existujícího řešení pro porovnání byl vybrán systém OpenAFS. Tento systém byl vybrán s ohledem na jeho rozšířenost jak v akademickém tak průmyslovém světě a také proto, že svým konceptem má ke KIVFS nejblíže a v základních principech z OpenAFS návrh KIVFS vychází. Testy byly provedeny na identickém zapojení tří serverů a jednoho klienta s konstantní rychlostí linek mezi servery 1Gbps. Linka mezi servery a klientem byla v jednotlivých krocích testu modifikována na hodnoty 10Mbps, 100Mbps a 1Gbps. Zapojení serverů a klienta je znázorněno na následujícím obrázku 7.12.



Obrázek 7.12: Model KIVFS zapojení testovacího prostředí

Jak bylo uvedeno v předchozích kapitolách, operace lze v distribuovaných souborech rozdělit na dvě množiny. První množinou operací je zpracování metadat, tedy vytváření a modifikace atributů souborů. Druhou množinou operací je zpracování obsahu souborů, tedy nahrání nebo stažení dat ze serverů. Při nahrání nového souboru na server se provádějí operace z obou množin. Při vykonávání

operací z každé z těchto množin, probíhá jiný typ komunikace a jednotlivé operace systém jiným způsobem zatěžuje. Při práci s metadaty je problematickou částí činnosti nutnost hlasování a tvorby logických hodin, tedy velmi intenzivní komunikace pomocí malých zpráv. Tento typ komunikace bude primárně ovlivněn hodnotami RTT. Nadruhou stranu nahrání nebo stažení dat je jiným druhem přenosu, neboť spojení je navázáno jen jednou a dále probíhá kontinuální datový přenos pomocí již navázaného spojení. Při přenosu velkých souborů bude RTT zpoždění zanedbatelné, ale naopak se při tomto typu přenosů projeví maximální dosažitelná přenosová rychlost. Pro ověření konkurence schopnosti KIVFS oproti existujícímu řešení OpenAFS bude nutné otestovat oba typy přenosů.

Testy byly realizovány jako postupné nahrávání deseti souborů o velikosti 10MB, 100MB a 1GB. Velikosti testovacích souborů byly zvoleny na základě zpracování statistik reálných dat na distribuovaném souborovém systému OpenAFS, provozovaném na KIV ZČU publikovaných v práci [75]. Každé z jednotlivých měření bylo opakováno desetkrát a ve výsledcích je uveden průměr z provedených měření. První test byl provedem na KIVFS v zapojení znázorněném na obrázku 7.12. Prostředí, v němž byl test proveden, bylo izolované od ostatních počítačových sítí tak, aby výsledek nebyl ovlivněn jiným provozem. Výsledné časy z prvního testu pro linky s přenosovou rychlostí 10Mbps jsou uvedeny v následující tabulce 7.10.

Velikost souboru	**OpenAFS** [**min**]	**OpenAFS** [**MBps**]	**KIVFS** [**min**]	**KIVFS** [**MBps**]
10MB	00:09.11	1.10	00:10.31	0.97
100MB	01:31.5	1.09	01:31.20	1.10
1000MB	15:25.78	1.08	14:53.39	1.12

Tabulka 7.10: Výsledky měření přenosu souborů v KIVFS a OpenAFS pro rychlost linek 10Mbps

Z výsledků je patrné, že KIVFS je se systémem OpenAFS, na pomalejších linkách s maximální přenosovou rychlostí do 10Mbps, srovnatelná. Maximální přenosová rychlost je pro oba systémy a pro jednotlivé velikosti souborů téměř shodná. U obou systémů je patrné, že je linka saturovaná a vyšších přenosových rychlostí není možné dosáhnout. Výsledek prvního testu je pozitivní, neboť nový systém není v základním nastavení kvalitativně horší než existující referenční rešení OpenAFS.

Druhý test byl proveden identicky jako první, ale na linkách s maximální přenosovou rychlostí 100Mbps. Výsledeky těchto měření jsou uvedeny v tabulce 7.11.

Na datových linkách s maximální přenosovou rychlostí 100Mbps už jsou mezi oběma systémy patrné rozdíly. Na malých souborech do 10MB je OpenAFS o 72% rychlejší. Toto zpoždění nového systému, které může na první pohled působit jako zásadní nedostatek, je způsobené režií hlasování o logických hodinách, které je nutné k realizaci multi-master RW Online replikace, jak bude uvedeno v části 9.3 a bude kompenzováno možností dynamického směrování, jak bude uvedeno v části 9. Naproti tomu na větších souborech nad 10MB, kdy je omezena manipulace s metadaty, je už nový systém výrazně výkonnější než referenční OpenAFS.

Velikost souboru	OpenAFS [min]	OpenAFS [MBps]	KIVFS [min]	KIVFS [MBps]
10MB	00:01.42	7.04	00:02.45	4.08
100MB	00:15.11	6.62	00:10.30	**9.71**
1000MB	02:37.16	6.36	01:31.04	**10.98**

Tabulka 7.11: Výsledky měření přenosu souborů v KIVFS a OpenAFS pro rychlost linek 100Mbps

Na souborech o velikosti 100MB KIVFS výkonější o 46% a na souborech o velikosti 1GB je maximální dosažená rychlost dokonce o 72% vyšší než u referenční OpenAFS řešení. Je evidentní, že nový distribuovaný souborový systém KIVFS je schopen efektivněji využívat kvalitnějších linek. Z výsledků testu na souborech o velikosti 1GB je patrné, že v případě KIVFS je i v tomto případě linka saturovaná a tedy není možné dosáhnout na této lince vyšších rychlosti. Aby se tato hypotéza ověřila, byl proveden třetí test na datových linkách s vyšší přenosovou rychlostí než je 100Mbps.

Třetí test byl proveden opět podle stejného scénáře jako oba testy předchozí, ale v systému byly instalované linky s přenosovou kapacitou 1Gbps, což je dnes běžný standard používaný v serverovém prostředí, ale částečně už i v uživatelském sektoru. Výsledky testů pro linky s rychlostí 1Gbps jsou uvedeny v tabulce 7.12.

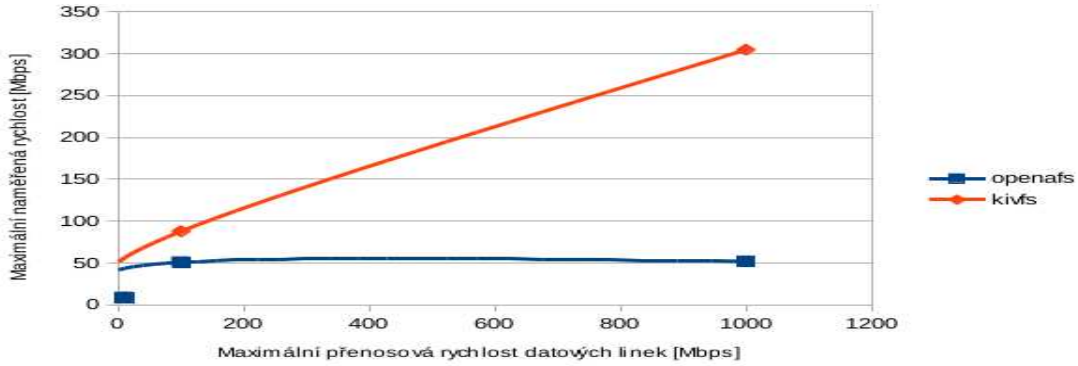
Vel. souboru	OpenAFS [min]	OpenAFS [MBps]	KIVFS [min]	KIVFS [MBps]
10MB	00:01.41	7.09	00:01.55	6.45
100MB	00:14.30	6.99	00:04.12	**24.27**
1000MB	02:34.36	6.48	00:26.31	**38.01**

Tabulka 7.12: Výsledky měření přenosu souborů v KIVFS a OpenAFS pro rychlost linek 1Gbps

Třetí test na linkách s rychlostí 1Gbps potvrdil a vylepšil výsledky druhého testu. Na malých souborech pod 10MB KIVFS za OpenAFS, díky již zmíněné režii hlasování, stále zaostává, ale tento rozdíl se s vyššími rychlostmi přenosových linek snižuje. Zatím co na linkách s maximální přenosovou rychlostí 100Mbps bylo OpenAFS o 72% rychlejší, na linkách s maximální přenosovou rychlostí 1Gbps už je tento rozdíl jen 9%. Výsledek testu na souborech o velikosti 10MB je tedy při změně linky s maximální přenosovou rychlosti z 100Mbps na 1Gbps o 63% lepší.

Ještě výraznější posun je možné pozorovat u testů na souborech o velikostech 100MB a 1GB. Při testu s 100MB soubory na lince o rychlosti 1Gbps je KIVFS již o 347% rychlejší oproti OpenAFS. U souboru o velikosti 1GB je KIVFS dokonce rychlejší 587%. U OpenAFS je patrné, ze posun maximální přenosové rychlosti linek z 100Mbps na 1Gbps znamená pouze 1.88% nárust. Je evidentní, že hranice maximální přenosové rychlosti je u OpenAFS na hranici 7MBps, tedy cca 56Mbps a vyšší maximální přenosová rychlost již tuto hodnotu neposune. Tento výsledek je dán implementací a s kvalitnějšími linkami s vyššími přenosovými rychlostmi již není možné maximální přenosovou rychlost OpenAFS posunout, a to i přesto,

ze v případě OpenAFS je použito výrazně jednodušších synchronizačních mechanizmů, díky použití master RW replikací. Naproti tomu u nového distribuovaného souborového systému KIVFS rychlost roste společně s maximální dostupnou přenosovou rychlostí. Tento rozdílný vývoj je dobře patrný na následujícím obrázku 7.13.



Obrázek 7.13: Porovnání vývoje maximální přenosové rychlosti OpenAFS a KIVFS ve vztahu ke změně maximální přenosové rychlosti datových linek

8. Multi-master RW Online replikace

Replikace dat je jednou z hlavních předností distribuovaných souborových systémů oproti lokálním nebo síťovým souborovým systémům a pro efektivní napojení mobilních zařízení do distribuovaného souborového systému, stejně jako pro provoz geograficky rozsáhlých systémů, má zcela zásadní význam, neboť zajišťuje dostupnost dat z více souborových serverů a tím snižuje pravděpodobnost nedostupnosti dat a také na základě existence více datových zdrojů pro jeden soubor, umožňuje volit optimální zdroj s ohledem na maximální dostupnou přenosovou rychlost, geografickou lokaci zdroje či cenu datové linky ke zdroji. Jak bylo popsáno v kapitole 5.3, existuje více možností jak replikace implementovat. Z hlediska fungování jsou podstatné dva aspekty:

- Možnost zápisu dat na více replik.
- Možnost zahájit replikaci dat ihned po aktualizaci či tvorbě dat.

Oba tyto aspekty jsou pro předpokládané použití na mobilních zařízeních či geograficky rozsáhlých systémech klíčové a u většiny systémů popsaných v kapitole 5.9 nastavené nevhodným způsobem či vůbec v daném systému neexistují jako možnost.

8.0.1 Multi-master RW replikace

Hlavním nedostatkem většiny existujících řešení, jak ukazuje tabulka 5.3, je používání *Master Write* kopie, kdy jsou data replikována na více uzlů, ale pouze na jednom je možné data modifikovat. Uzel, který je použit pro zápis je navíc, například u OpenAFS, volen staticky administrátorem systému. Toto řešení je výhodné z hlediska jednoduché implementace a administrace, neboť chování takových systémů je identické s centralizovanými datovými úložišti. Ale nutnost používat jedinou repliku pro zápis je nevhodné pro mobilní zařízení stejně jako pro geograficky rozsáhlé systémy, neboť klienti mají k různým souborovým serverům různě rychlý a různě stabilní přístup, jak bylo popsáno v kapitole 9.2. KIVFS proto implementuje metodu *Multi-master RW replikace*. V modelu používajícím *Multi-master RW replikaci* mohou být data zapsána na libovolný uzel. Výběr vhodného uzlu se provádí na základě hledání optimálního uzlu z kapitoly 9.2 a bude detailně popsán v 9. Při použití *Multi-master RW replikace* se mohou data měnit na libovolném uzlu, je tedy nutné zajistit dodržování zvoleného konzistenčního modelu. V KIVFS je používán model konzistence, kde nemusí být na všech replikách zcela aktuální data, ale klient který poptává data je dostane vždy aktuální. Tento model konzistence je v KIVFS řešený pomocí uzamykání jednotlivých replik souborů na úrovni metadat. Každý soubor má v rámci *metadat* v databázi uloženy informaci o aktuální verzi. Tato informace je celočíslený čítač, který je při vytvoření souboru nastaven na hodnotu *1* a při změnách dat inkrementován o hodnotu *1* s každou provedenou operací zápisu. Pokud klient nahrává nová data, je nahrátí provedeno podle následujícího scénáře:

1. Uzamčení souboru a všech jeho replik prostřednictvím DB modulu, od tohoto okamžiku nemůže data měnit nikdo jiný. Tento zámek je platný v celém systému.

2. Zahájení nahrávání dat souboru na souborový server, na pozadí probíhá online defragmentace a volitelné šifrování a zahájení *online replikace*, které popisuje následující kapitola 8.1.

3. Po dokončení nahrávání dat, proběhne inkrementace identifikátoru verze u zpracovávaného souboru, ale jen pro repliku na souborovém serveru kam jsou data klientem nahrávána.

4. Postupná aktualizace identifikátoru verze, a tedy uvolňování souborů pro klienty, podle dokončování replikací na jednotlivých dalších souborových serverech.

Samozřejmě může, a v distribuovaném prostředí, kde se používají mobilní zařízení obzvláště, nastat situace, kdy klient není schopen z důvodu přerušení konektivity nahrávání dat dokončit. S touto situací je třeba počítat, neboť pokud by nebyl tento stav ošetřen, přerušení nahrávání dat by způsobilo trvalou nedostupnost dat, neboť by tato zůstala trvale uzamčena. Klasicky se tento problém řeší pomocí timeoutu, tedy pokud operace neproběhne v maximálním povoleném čase, je ukončena. Bohužel toto řešení, byť implementačně jednoduché, může v geograficky rozsáhlých systémech v kombinaci s mobilními formami připojení způsobovat značné komplikace, neboť nahrávání velkých dat, řádově stovky megabytů nebo gigabytů, může na mobilním zařízení trvat hodiny nebo dokonce i dny. V takovém případě by docházelo k dvěma nevhodným situacím s ohledem na nastavení hodnoty timeout:

- Vysoká hodnota timeoutu: Při přerušení nahrávání dat bude systém ještě velice dlouho a zbytečně čekat, než prohlásí spojení za ukončené a operaci přeruší a data odemkne. Navíc při vysokých, ale stále realistických hodnotách, jako například 10 hodin, stále může docházet k násilnému ukončení spojení před dokončením nahrávání dat.

- Nízká hodnota timeoutu: Při nízké hodnotě systém rychleji pozná přerušené spojení, ale na druhou stranu může násilně ukončovat dlouhá spojení, která vzniknout nahráváním velkých dat po pomalejších linkách mobilního připojení.

S ohledem na nutnost zavést ukončovací podmínky a nevhodnost použití timeoutu na délku spojení a s přihlédnutím ke skutečnosti, že KIVFS komunikuje protokolem založeným nad TCP, bylo rozhodnuto že ukončovací podmínkou je odpojení klienta od Auth modulu. TCP si samo udržuje informaci, zda je spojení ještě aktivní a pokud během trvání spojení klient dokáže v rámci TCP protokolu udržovat spojení, je možné předpokládat, že bude tímto spojením stále schopen posílat data, byť přenosová rychlost může být velice malá.

Pokud v průběhu aktualizace souboru vznikne od jiného klienta požadavek na zápis dat do stejného souboru, je tento požadavek zařazen do fronty pro zpracování a proveden až po dokončení probíhajícího zápisu. V okamžiku dokončení uploadu souboru klientem nejsou aktuálně nahratá data dostupná ze všech replik.

Synchronizace replik probíhá paralelně a na pozadí, jak bude uvedeno v následující podkapitole 8.1. Aby bylo možné dosáhnout silné konzistence replikovaných dat, nejsou klientovi předána požadovaná data ze souborového serveru, ke kterému je připojen, ale z takového serveru, který obsahuje verzi dat s nejvyšším číslem revize. Tím, že jsou *metadata* online a synchronně replikována do celého systému, má každý z uzlů aktuální kopii kompletních metadat a na číslo revize se dotazuje pouze do lokální kopie databáze, což výrazně snižuje nutný čas k dohledání aktuální kopie. Číslo revize není použité jen při požadavku čtení či zápisu konkrétních dat, ale také k udržování aktuálního stavu cache klienta, která drží část obsahu souborů. Tím jak jsou data v distribuovaném prostředí modifikována, dochází k zastarávání údajů v klientských cache. Proto se klienti, před každým předáním dat uživateli a delší době nečinnosti, dotazují na aktuální revize dat, která mají v cache a v případě, že v KIVFS již existuje vyšší číslo revize pro požadovaná data než má klient, jsou data z cache odstraněna a nahrazena aktuálnější kopii. Toto vše probíhá na pozadí systému bez vědomí a nutnosti zásahu uživatele či administrátora.

8.1 On-line replikace

Další podstatnou nevýhodou běžně používaných modelů replikace pro geograficky rozsáhlé a kvalitou sítě různorodé systémy je zpoždění replikace. V systémech jako je OpenAFS, je replikaci dat, která je navíc často jen v režimu read-only, nutné spouštět manuálně. Navíc replikaci smí spouštět pouze administrátor systému. Tento postup je vhodný maximálně pro některá data jako jsou obrázky pro webové stránky nebo obecně data, kde nenastává častá změna. Pokud by ale měl distribuovaný souborový systém dobře sloužit pro jakékoli typy dat, kde může ke změnám docházet často a nedeterministicky, je tento nejjednodušší model nevhodný. Výše uvedený problém je možné částečně eliminovat cyklickým spouštěním replikace pomocí plánovače úloh operačního systému. To už je efektivnější metoda, ale je stále nevhodná pro systémy, kde chceme umožnit klientům přístup k nejvýhodnější replice dat z pohledu přenosových parametrů sítě. Pokud by se replikace spouštěla cyklicky, budou existovat okamžiky, kdy všechny repliky nebudou mít aktuální data a pro zachování silné konzistence bude nutné požadavky všech klientů posílat na jedinou repliku s aktualizovanými daty. Tento problém je, že budou vznikat situace, kdy systém bude replikaci spouštět zbytečně, neboť k žádné změně dat nedošlo. Druhý problém by šlo úspěšně eliminovat kontrolou verzí metadat, ale i tato operace by dočasně zatěžovala systémové zdroje, aniž by byla tato operace potřebná.

Na základě výše uvedených nedostatků, byl pro KIVFS rozšířen model *Multimater RW replikace* o možnost započít replikaci automaticky a okamžitě jakmile jsou data změněna. Tento model replikace budeme dále označovat jako *Multimaster RW On-line replikace*. Cílem *Multi-master RW On-line replikace* je minimalizovat zpoždění distribuce replikovaných dat a tím zvýšit dostupnost alternativních zdrojů identických dat pro klienty. V KIVFS je replikační proces zahájen v okamžiku, kdy je dokončeno nahrání prvního celého fragmentu modifikovaných dat. Ukládání dat po jednotlivých fragmentech je popsáno v kapitole 7.7.2. U souborů, jejichž velikost nepřesáhne konfiguračně nastavenou velikost jednoho fragmentu, je replikace spouštěna po nahrání poslední části modifikovaných dat.

Replikace je spouštěna v rámci FS modulu, který podle nastavení cílů replikace pro daný svazek, naváže spojení s ostatními souborovými servery a začne odesílat nová data. Informace o nastavení replikací je uloženo v databázi a je nastavované administrátorem pro jednotlivé svazky. Pro každý souborový server, na který jsou odesílána nová data, je spuštěno samostatné vlákno, díky čemuž probíhá replikace paralelně.

Ihned po dokončení přenosu replikovaných dat na každý jeden souborový server je v *metadatech* aktualizována datová revize a soubor je uvolněn pro přístup klientů. Navržená *Multi-master RW On-line replikace* po souborových fragmentech výrazně snižujeme replikační zpoždění, neboť potřebný čas na dokončení sychronizace všech replik je jen tak dlouhý, jak dlouho trvá replikace jednoho celého fragmentu souborů. Díky tomu je možné rychleji poskytnout klientům více alternativních datových zdrojů pro replikovaný soubor, což se s výhodou využije při implementaci *dynamického směrování požadavků*, které je popsané v následující kapitole 9. Časová úspora při replikaci dat narůstá s velikostí replikovaného souboru, jak ukazuje tabulka 8.1. Jak je z tabulky 8.1 patrné, je pro větší

Způsob replikace	**1MB**	**100MB**	**1000MB**
Cyklická 3600s pro 10Mbps	3600s	3650s	3900s
Cyklická 600s pro 100Mbps	601s	620s	700s
Cyklická 600s pro 1Gbps	601s	610s	620s
On-line 10Mbps	5s	10s	100s
On-line 100Mbps	4s	4s	80s
On-line 1Gbps	3s	4s	12s

Tabulka 8.1: Porovnání zpoždění replikačních modelů

soubory *Multi-master RW On-line replikace* výrazně rychlejší. Když porovnáme měření 1GB souboru na 10Mbps lince, je rozdíl cyklické replikace a navrženého modelu *Multi-master RW On-line replikace* 6.3x rychlejší. Tato výhoda bude ještě výraznější v případě nižších přenosových rychlostí. U souborů větších než je dvojnásobek zvolené velikosti fragmentu souborů bude replikační zpoždění rovno maximálně dvojnásobku času nutného k přenesení jednoho replikovaného fragmentu, neboť replikace prvního bloku začíná ihned po jednom nahrátí na první souborový server.

8.2 Výhody plynoucí z Multi-master RW On-line replikace

Navržený model *Multi-master RW On-line replikací* přináší do KIVFS následující výhody :

- Možnost aktualizovat data na libovolném souborovém serveru, díky verzování datových souborů a zvoleného modelu *silné konzistence*
- Minimální zpoždění při synchronizaci replik díky fragmentaci a spuštění replikace již v průběhu nahrávání dat

- Minimalizace zátěže datových linek, neboť k replikačnímu přenosu dochází při změně dat jednoho konkrétního souboru či fragmentu a nikoliv v pevně daný čas či na základě požadavku administrátora.

- Rozklad zátěže datových linek mezi servery, neboť při replikaci je, stejně jako při výměně dat s klientem, použito dynamické směrování požadavků, jak bude popsáno v kapitole 9.3, díky čemuž systém může v krátkém čase, který je dán časem replikace jednoho fragmentu souboru, měnit směrování datových požadavků a tak optimalizovat datové toky replikovaných dat.

Už samostatně je benefit zavedení *Multi-master RW On-line replikace* značný, jak je vidět z předchodzí tabulky 8.1. Význam zavedení Multi-master RW On-line replikací nebyl motivován pouze rychlým dosažením synchronního stavu replikací, ale hlavně byl zamýšlen jako podkladová platforma pro možnost dynamického směrování požadavků, které bude popsáno v následující kapitole 9.

9. Dynamické směrování požadavků v rámci KIVFS

Nasazení distribuovaného souborového systému v požadované oblasti mobilních zařízení či geograficky rozsáhlých systémů má jeden společný problém a tím jsou parametry přenosových médií, ať už nízká přenosová rychlost nebo velké zpoždění linky při přenosu dat na velké vzdálenosti. Pro možnost praktického nasazení distribuovaného souborového systému na mobilní zařízení či geograficky rozsáhlý systém je nutné optimalizovat datové cesty. S parametry sítě nejsme schopni udělat nic, ale můžeme optimalizovat datovou cestu tak, aby využívala výhodnější spoje před přímými spoji. V prostředí počítačových sítí se k tomuto účelu používají dynamické směrovací protokoly popsané v kapitole 4. Tyto směrovací protokoly berou počítačovou síť jako neorientovaný ohodnocený graf a v něm hledají minimální cestu umožňující přenášet maximální datový tok. Použití klasických dynamických směrovacích protokolů, popsaných v kapitole 4 není v našem případě možné, neboť se nejedná o nasazení v jedné sítí, v rámci které můžeme parametry a nastavení sítě libovolně měnit. Datové cesty geograficky rozsáhlých systémů typicky využívají sítě Internet a prochází tedy sítěmi mnoha ISP, stejně tak mobilní zařízení se připojují k aktuálně dostupné konektivitě bez ohledu na její příslušnost k jedné či více organizacím.

Jelikož je v KIVFS implementovám systém *Multi-Master RW On-line replikace*, jak bylo uvedeno v předchozí kapitole 9.3, máme možnost ke stejným datům přistupovat prostřednictvím více souborových serverů a tedy po různých datových cestách, v rámci kterých si můžeme vybrat nejvýhodnější variantu. Navíc jednotlivé uzly KIVFS znají všechny uzly ostatní a mohou komunikovat každý s každým. Tím nám nad počítačovou sítí vzniká nová překryvná síť tvořená uzly distribuovaného souborového systému a můžeme dynamické směrování realizovat na aplikační úrovni a tím docílit podobných výhod jako mají dynamické směrovací protokoly, ale bez nutnosti mít kontrolu nad všemi fyzickými linkami a směrovacími zařízeními počítačové sítě.

9.1 Automatická konfigurace klienta

Klient klasického distribuovaného souborového systému, jako je například NFS4, musí znát alespoň jeden server distribuovaného systému, ke kterému se připojuje. To je nejjednodušší a často používaná konfigurace. Pro pokročilejší systémy, jako je například OpenAFS, je možné zadat serverů více, aby systém fungoval i po výpadku jednoho nebo více serverů. Tedy pokud aktuálně používaný server není k dispozici, systém se přepojí na další v pořadí. Prioritu či pořadí serverů určuje administrátor, ať už na straně klienta nebo na straně serveru, volbou preference. Tato preference může být navíc geograficky vázaná a tak v různých lokalitách může být preferován jiný server bez nutnosti rekonfigurace klienta. Tento model konfigurace byl dosud dostačující. V dnešní době, kdy není možné jednoduše predikovat místo připojení klienta, neboť klient během jediného dne může změnit lokaci i o tisíce nebo desítky tisíc kilometrů a stejně tak není možně predikovat chování sítě, která díky násobným spojům na úrovni mezinárodních, ale dnes

už i národních ISP, dynamický mění směrování podle aktuální situace, je tento klasický model přežitý. Aby klient, kterému chceme poskytnout možnost plné mobility, mohl k datům geograficky rozsáhlých systémů přistupovat efektivně odkudkoliv, musí být klasický model upraven.

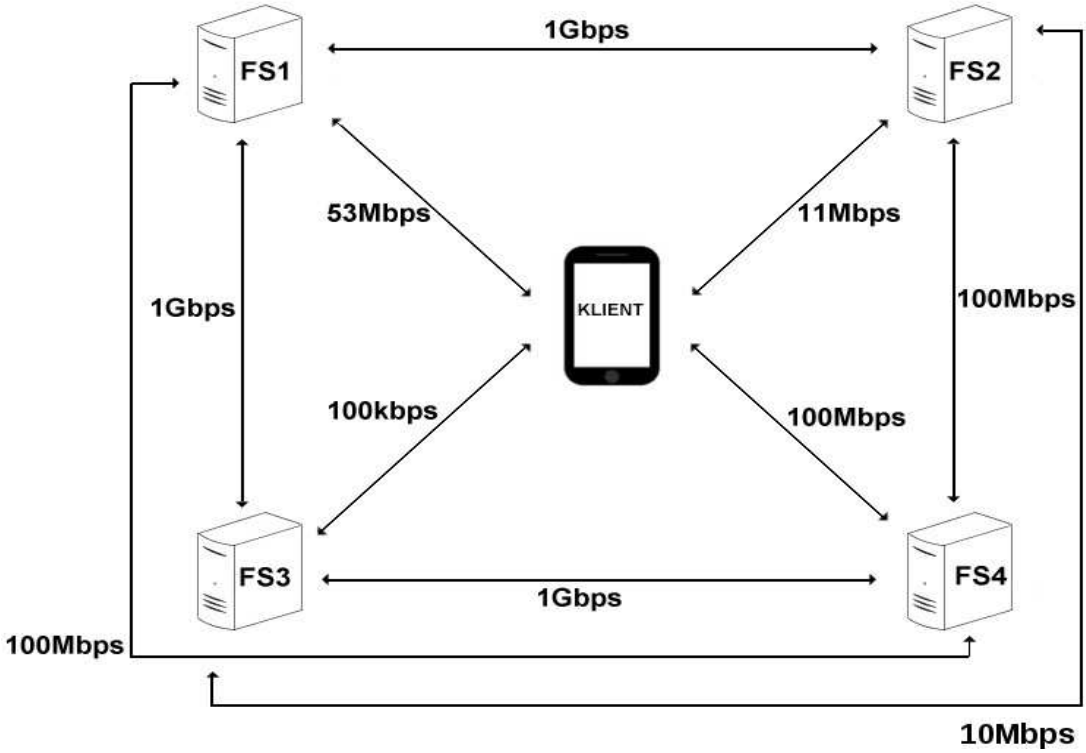
V případě klienta je situace velmi jednoduchá a dynamické směrování zde bude použitou pouze okrajově. Klient v KIVFS má v konfiguraci k dipozici kompletní sadu všech dostupných serverů, kteří poskytují Auth modul v rámci distribuovaného sourového systému KIVFS. Pro připojení v prvním kroku použije první server v pořadí bez ohledu na vlastní lokaci či lokaci serveru. Pokud se mu nepodaří spojit se v čase definovaném timeoutem TCP spojení, což je ve výchozím stavu 30s, použije v pořadí další server z konfigurace. Pokud se připojení a ověření podaří, může klient začít okamžitě se systémem pracovat.

Po úspěšném připojení a ověření klienta do KIVFS, je na pozadí na klientovi spuštěno testovací vlákno, podobně jako je tomu mezi jednotlivými servery, jak je popsáno v kapitole 9.2. Klient tak získá infomaci o kvalitě linek ke všem dostupným serverům a na základě této tabulky provede přepojení k serveru s linkou, která v rámci měření dopadla nejlépe. Tento postup testování a případně přepojení je prováděn opakovaně, neboť je nutné přepokládat, že parametry sítě se v čase mění. Testování a přepojení je prováděno opakovaně s periodou nastavenou v konfiguračním souboru, s výjimkou prvního testování, které je provedeno ihned po připojení a ověření klienta do systému. První testování až po autorizaci a autentizaci klienta bylo zvoleno záměrně a to z důvodu zachování jediného přístupového bodu do celého systému, ale také s ohledem na bezpečnost, aby nemohlo docházet k pokusům o DOS[97] nebo DDOS[98] útok, který by byl veden od neautorizovaných klientů. V tuto chvíli Auth modul na straně serveru jako první po připojení očekává provedení autorizace a autentizace, jak je popsáno v kapitole 7.3 a při jakémkoli pokusu o jinou komunikaci ukončuje spojení. Pokud se pokusy o neautorizovaný přístup opakují, je možné jim zamezit pomocí nástrojů jako je fail2ban, kde tento nástroj případnému útočníkovi po N neúspěšných pokusech o připojení zablokuje přístup pomocí firewallu natrvalo nebo na pevně stanovenou dobu. Tunelované spojení, které je popsané v následující kapitole 9.2 není v tomto případě možné ani nutné použít, neboť se dá předpokládat, že linky mezi servery budou vždy kvalitnější než linky klienta k jednotlivým serverů. Na straně klienta požadujeme linky s nejlepšími parametry bez ohledu na vše ostatní.

9.2 Tunelování spojení mezi servery

Jak bylo popsáno v kapitole 7.7, pokud klient požaduje nějaká data, jsou mu po jejich nalezení v DB modulu, vráceny KIVFS protokolem parametry spojení na FS server, který pro každý jednotlivý přenos dat připraví samostatné spojení. Tento přenos je vázaný na skutečnou lokaci dat. Pokud budeme mít čtyři servery FS1, FS2, FS3 a FS4 a zapojení z následujícího obrázku 9.1 a klientem požadovaná data budou uložena jen na serveru FS2, byl by přenos proveden mezi klientem a serverem FS2, bez ohledu nato, k jakému serveru je klient dle informací z předchozí kapitoly 9.1 připojen. To je pochopitelné, protože server FS1, který je pro klienta nejvýhodnější z pohledu parametrů linky, neboť se jedná 100Mbps linku, což je z dostupných linek k ostatním serverům nejlepší možná. Ale server FS1 neobsahuje požadovaná data. To je ale velice nevýhodné, neboť data se ke klientovi

budou přenášet po lince, která má maximální přenosovou rychlost 11Mbps a je tedy téměř 10x pomalejší než linka k serveru FS1.



Obrázek 9.1: Model zapojení KIVFS pro příklad tunelového spojení pomocí dynamického směrovacího protokolu

Tento zjevný nedostatnek je v KIVFS řešen pomocí dynamického směrování požadavků a tunelování linek. Server FS1 přijme od klienta požadavek na přenos dat, která jsou uložena jen na serveru FS2, ale nekontaktuje přímo server FS2 s žádostí o přípravu přenosu. Server FS2 na úrovní FS modulu projde směrovací tabulku dostupnou z Sync modulu, jejíchž tvorba je popsána v kapitole a z tabulky vybere optimální cestu k FS2. Což v našem případě dle obrázku 9.1 nebude příme spojení klient a FS2, ale bude to cesta **klient,FS4,FS3,FS1,FS2**. Postupně budou kontaktovány jednotlivé servery pro sestavení **tunelu** pro přenos dat. Po jeho sestavení, dostane klient parametry pro přenos souboru, kde přípojným místem pro něj bude nejvýhodnější server z jeho pohledu a to je v tomto případě FS4 a celý přenos po uvedené cestě bude, s ohledem na nejpomalejší linku v cestě, realizován 100Mbps rychlostí, namísto původních 11Mbps v případě přímého spojení na FS2.

Z obrázku 9.1 je patrné, ze 100Mbps spojení je možné realizovat i kratší cestou z FS1 na FS2, kde linka mezi těmito servery má také maximální přenosovou rychlost 100Mbps. Toto spojení ale nebude použito a bude preferována rychlejší cesta, byť realizovaná přes více serverů. Důvodem je snaha o dosažení maximální propustnosti v systému a zatímco přenos mezi FS1 a FS2 by tuto linku saturoval, přenos mezi FS1 a FS3, FS3 a FS1 a FS1 mezi FS2 je realizován po linkách s maximální přenosovou rychlostí 1Gbps a tedy linky budou využity jen z cca 10% a bude možné těmito linkami realizovat souběžně i další přenosy bez snižování přenosové rychlosti při paralelních přenosech. Pochopitelně tímto postupem se

linka, při větším počtu přenosů, začně postupně saturovat a tím i snižovat svoji maximální přenosovou rychlost. To ale nevadí, neboť směrovací tabulka je cyklicky přepočítávána, jak je uvedeno v kapitole 9.2 a díky tomu dojde postupně k přelévání provozu na další volné linky a tím k rozkladu provozu.

Na příkladu zapojení systému, který je znázorněn na obrázku 9.1, je možné provést stejné testy jako v kapitole 7.9. Cílem opakování testů s použitím dynamického směrovacího protokolu je zjistit, jak moc budou tunelovaná spojení negativně ovlivňovat maximální přenosovou rychlost. Výsledky opakovaných měření z kapitoly 7.9 jsou znázorněny v následujících třech tabulkách 9.1, 9.2, 9.3.

Velikost souboru	**OpenAFS [min]**	**OpenAFS [MBps]**	**KIVFS [min]**	**KIVFS [MBps]**
10MB	00:09.11	1.10	00:10.52	0.95
100MB	01:31.5	1.09	01:31.20	1.09
1000MB	15:25.78	1.08	14:53.39	1.12

Tabulka 9.1: Výsledky měření přenosu souborů v KIVFS a OpenAFS pro rychlost linek 10Mbps s využitím dynamického směrování požadavků pro KIVFS

Velikost souboru	**OpenAFS [min]**	**OpenAFS [MBps]**	**KIVFS [min]**	**KIVFS [MBps]**
10MB	00:01.42	7.04	00:02.46	4.07
100MB	00:15.11	6.62	00:10.53	**9.50**
1000MB	02:37.16	6.36	01:31.06	**10.98**

Tabulka 9.2: Výsledky měření přenosu souborů v KIVFS a OpenAFS pro rychlost linek 100Mbps s využitím dynamického směrování požadavků pro KIVFS

Velikost souboru	**OpenAFS [min]**	**OpenAFS [MBps]**	**KIVFS [min]**	**KIVFS [MBps]**
10MB	00:01.41	7.09	00:01.59	**6.29**
100MB	00:14.30	6.99	00:04.45	22.47
1000MB	02:34.36	6.48	00:27.00	**37.04**

Tabulka 9.3: Výsledky měření přenosu souborů v KIVFS a OpenAFS pro rychlost linek 1000Mbps s využitím dynamického směrování požadavků pro KIVFS

Na všech naměřených výsledcích je patrné, že při použití dynamického směrování nedojde k výraznému zhoršení přenosových parametrů systému. Nejzásadnější výsledky jsou pro nás měření na 1Gbps lince pro 10MB a 1000MB soubory, neboť se jedná o nejčastější kombinace, které jsou na serverech dostupné. V obou těchto případech dynamické směrování v rámci KIVFS umožnilo dosáhnout přenosové rychlosti jen o cca 2% horší než je maximální možná přenosová rychlost

KIVFS systému, ale s využitím dynamického směrování, tedy s možností reagovat dynamicky na změny v síti. Uvedený rozdíl 2% je dán režií dynamického směrování a sestavování tunelů. Sice to není zcela zanedbatelná režie, ale s přihlédnutím k benefitu v podobě posunu od 1.12 MBps pro 10Mbps k 37.04MBps pro 1Gbps linku a pro 1GB soubor a 1Gbps linky mezi servery, je toto zpozdění akceptovatelné. Díky provedeným testům se podařilo prokázat přínosnost dynamického směrovaní požadavků v novém distribuovaném souborovém systému KIVFS.

9.3 Replikace dat s využitím dynamického směrování

Přenosy dat mezi klientem a servery nejsou jediným datovým přenosem, který se v rámci KIVFS provádí. Jak bylo uvedeno v kapitole 7.7.5 a jsou data při nahrávání na server téměř okamžitě online replikována na další servery. I tento datový přenos je možné optimalizovat s využitím dynamického směrování, neboť stejně jako v případě přímé datové cesty mezi klientem a serverem, existují i mezi servery spoje s různými parametry a s různým zatížením a přímá linka mezi servery nemusí být nutně ta nejvýhodnější. V případě KIVFS se i na replikovaná spojení uplatňuje princip dynamického směrování požadavků, který zajistí přenos dat nejvýhodnější datovou cestou a to i s ohledem na změny parametrů, které v síti nedeterministicky nastavávají. Systém dynamického směrování se uplatňuje samostatně pro každý realizovaný přenos. Je zřejmé, že pro efektivní fungování bude výhodnější realizovat více kratších datových přenosů než méně dlouho trvajících spojení, neboť pokud se parametry linky změní v průběhu přenosu, nebyl by systém schopen reagovat dříve, než by byl celý přenos dokončen. Tento požadavek je v KIVFS splněn díky fragmentaci dat popsaných v kapitole 7.7.2, neboť přenosy velých souvislých datových bloků jsou realizovány po jednotlivých fragmentech a systém má možnost provést přenos každého jednotlivého fragmentu jinou datovou cestou a tím efektivně reagovat na nedeterministické změny v síti.

10. Závěr

Cíle, které byly stanoveny v kapitole 6 považujeme za splněné a splnění prokázané na základě provedených testů. Výsledkem práce je nový distribuovaný souborový systém, který lze, díky Multi-master RW Online replikacím a dynamickému směrování požadavků, nasadit jak na mobilní zařízení tak pro geograficky rozsáhlé systémy a splňuje tak všechny požadované vlastnosti distribuovaného souborového systému definované v tabulce 5.3, čímž byl splněn cíl číslo 1.

KIVFS má navržen a implementován mechanizmus Multi-master RW on-line replikací [103]. Data nejsou ukládána jako jednotlivé soubory, tak jak přicházejí od klienta, ale jsou v průběhu ukládání dělena na fragmenty o pevné velikosti. Replikace nezačne po dokončení nahrávání celého souboru, ale po dokončení nahrávání jednoho fragmentu. Celková doba pro dokončení replikace jednoho souboru je tedy maximálně tak dlouhá, jako je doba nutná k replikaci jednoho bloku dat, čímž byl splněn cíl číslo 2.

Pro splnění cíle číslo 3 byla nad reálnou sítí, kterou nejsme schopni ovlivnit, vytvořena virtuální překryvná síť[102]. Při realizaci datových přenosů nebudou spoje v KIVFS nutně realizované přímo, ale s ohledem na hodnoty ve směrovací tabulce KIVFS budou tam, kde to bude výhodné, tunelovány přes další servery. Tento princip dovoluje přenos realizovat maximální možnou přenosovou rychlostí, neboť z dostupných linek vybere ty s nejvýhodnějšími parametry, ať už z hlediska maximální přenosové rychlosti nebo z hlediska RTT zpoždění, což je výhodné pro přenosy v geograficky rozsáhlých systémech. Návrhem a implementací překryvné sítě, dynamického směrování a tunelovaných spojení byl splněn cíl číslo 3.

Implementace KIVFS byla provedena v rámci bakalářských [68],[69],[70],[71] a diplomových prací[72],[73],[74]. Z možností KIVFS čerpala disertační práce, jejíchž tématem byla optimalizace cache na straně klienta[75],[76], [77]. Aktuálně ze základů KIVFS čerpají další dvě disertační práce. První práce zkoumá možnosti predikce datových toků v distribuovaném souborovém systému s využitím překryvných sítí [78],[79], druhá se zabývá možností propojení distribuovaného souborového sytému a hierarchického souborového systému, s využitím dynamického směrování a tunelování datových spojení. Kromě disertačních prací, na KIVFS navazují i další bakalářské a diplomové práce, jako například tvorba klienta pro mobilní platformu iOS společnosti Apple, která využívá již realizovanou HTTP proxy [74].

V současné době probíhá nad KIVFS další výzkum a to hned v několika směrech. První směr rozvoje řeší problematiku fungování logických hodin. Během testů se zjistilo, že celková výkonnost systému je zcela zásadním způsobem ovlivněna rychlostí a efektivitou fungování logických hodin. V současné verzi KIVFS jsou spoje realizující synchronizaci logických hodin realizovány přímým spojeními mezi servery. Dalším krokem výzkumu je ověření možnosti realizovat i tato spojení s využitím dynamického směrování požadavků. Druhým započatým směrem výzkumu je optimalizace uložení a cachování dat. Rychlost systému, především na serverové straně, není ovlivněna jen kvalitou přenosových linek, ale i rychlostí datových úložišť samotných. V rámci druhého směru výzkumu nad KIVFS se řeší možnost propojení distribuovaného souborového systému KIVFS, hierarchického modelu uložení dat a dynamického směrování požadavků. Cílem tohoto výzkumu

je navrhnout optimalizovaný model automatické migrace dat v rámci serverových datových úložišť s ohledem na zvýšení rychlosti přístupu k nejčastěji používaným datům.

Název práce: Dynamické směrování v distribuovaných souborových systémech

Autor: Luboš Matějka

Katedra: Katedra informatiky a výpočetní techniky

Vedoucí diplomové práce: prof. Ing. Jiří Šafařík, CSc., KIV FAV ZČU

Abstrakt: Práce popisuje nový distribuovaný systém KIVFS, jeho jednotlivé moduly a vlastnosti. Dále se práce věnuje problému propojení distribuovaného souborového systému a mobilních zařízení. Na základě zjištěných úskalí, jakou jsou omezené datové a energetické zdroje či nestabilní a různorodé datové připojení mobilních zařízení jsou navrženy dvě inovace distribuovaného souborového systému. První inovací je zavedení dynamického směrování požadavků uvnitř distribuovaného souborového systému. Druhou inovací je implementace Multi-Master RW Online replikace, která dynamické směrování zefektivní. Dopady obou inovací na výkon distribuovaného souborového systému jsou experimentálně ověřeny v distribuovaném souborovém systému KIVFS.

Klíčová slova: distribuovaný souborový systém, replikace, směrování

Title: Dynamic routing in distributed file system

Author: Luboš Matějka

Department: Department of Computer Science and Engineering

Supervisor: prof. Ing. Jiří Šafařík, CSc., KIV FAV ZČU

Abstract: The thesis describes new distributed file system KIVFS, its modules and attributes. Further, the thesis deals with the problem of interconnecting of distributed file system and mobile devices. Based on found limitations as are restricted data and energy sources or unstable and various data connections of mobile devices, two innovations of distributed file system are proposed. First one is introduction of dynamic routing of requirements inside of the distributed file system. Second one is implementation of Multi-Master RW Online replication which increases dynamic routing efficiency. Impact of both innovations on distributed file system performance is verified experimentally in distributed file system KIVFS.

Keywords: distributed file system, replication, routing

Aktivity autora

Publikace související s disertací

Pešička L., Matějka L., Racek S., Šafařík J., *Performance improvement of distributed file system using tunneling*, Přijato na 5th European Conference on the Engineering of Computer Based Systems, Larmaca 2017.

Skupa J., Matějka L., Šafařík J., *Dynamic Internal Message Routing in Distributed File System*, 2015 IEEE 13th International Scientific Conference on Informatics, pp. 236-240.

Matějka L., SKupa J., Strejc R., Pešička L., Šafařík J., *Dynamic Routing in Distributed File System*, Journal of Networks, 2014, vol. 14, pp. 251-259.

Bžoch P., Matějka L., Pešička L., Šafařík J., *Towards Caching Algorithm Applicable to Mobile Clients*, 2012 Federated Conference on Computer Science and Information Systems (FedCSIS), Wroclaw, 2012, pp. 607-614.
Citováno v :

> Dr. Bharat Singh Deora, Sushma Satpute, *Architecture of cloud server with cache on server*, Second International Conference on Information and Communication Technology for Competitive Strategies 2016, no. 14, pp. 5.
> Iyad Ollite, Dr. Nawaz Mohamudally, *Cost Efficient RESTful Service Caching for Mobile Devices*, International Journal of Software and Web Sciences 2015, pp. 96-101.
> I. Ollite, N, Mohamudally, *Performance analysis of a 2-tier caching proxy server for mobile RESTful services*, International Conference an Computer as a Tool, Salamanca 2015, pp. 1-7.

Bžoch P., Matějka L., Pešička L., Šafařík J., *Design and Implementation of a Caching Algorithm Applicable to Mobile Clients*, Informatica, 2012, vol. 36, no. 4.

Matějka L., Pešička L., Šafařík J., *Distributed file system with online multi-master replicas*, 2011 Second Eastern European Regional Conference on the Engineering of Computer Based Systems, Bratislava, 2011, pp. 13-18.
Citováno v:

> Shiva Asadianfam, Mahboubeh Shamsi, Shahrad Kashany, *A review: Distributed File System*, Iternational Journal of Computer Network and Communications Security, 2015, vol. 3, no. 5, pp. 229-234 .

Matějka L., Junák M., Pešička L., Pivnička M., Skupa J., Strejc R., Steiner V., Šafařík J., *KIV-DFS-Experimental Distributed File System*, Informatics 2009, Košice: Technical University, 2009, pp. 45-50.

Matějka, L., Pešička, L., Šafařík, J., *Dynamic routing in distributed fi-*

le system for mobile devices, Proceedings of CSE 2008 International Scientific Conference on Computer Science and Engineering. Košice: Technical University, 2008. pp. 263-268.

Matějka L., Šafařík J., *Distributed file systems on mobile devices*, Proceedings of the seventh international scientific conference Electronic computers and informatics ECI 2006. Košice: Vienala Press, 2006. pp. 42-47.

Matějka L., *Distribuované souborové systémy*, Počítačové architektury & diagnostika. Praha: České vysoké učení technické, 2005. pp. 125-128.

Publikace ostatní

Girg P., Matějka L., Pešička L., Skupa J., Šafařík J., *Parallel Computing with SMP in GNU/Linux*, Informatics 2009. Košice: Technical university, 2009. s. 57-62.

Otta J., Pešička L., Matějka L., Girg P., *Scientific Computing to Your Office and Your Home*, Champaign, IL, USA, 2007

Projekty

Otta J., Pešička L., Matějka L., Girg P.: Webmath.zcu.cz

Výuka

Roky	**Předmět**	**Rozsah**
2004-2006	PPA1 - Počítače a programování 1	cvičení
2004-2006	PPA2 - Počítače a programování 2	cvičení
2010-2014	UPS - Úvod do počítačových sítí	cvičení
2013	PC - Programování v jazyce C	cvičení
2009-dosud	SPOS - Správa serverů a počítačových sítí	přednásky, cvičení
2016-dosud	BSA - Bezpečnost síťových architektur	přednásky, cvičení

Literatura

[1] Ben Morris *The Symbian OS Architecture Sourcebook: Design and Evolution of a Mobile Phone OS.* Wiley; 2007.

[2] Mark L. Murphy *Beginning Android 4.* Wrox, 2012.

[3] Jonathan Levin *Mac OS X and iOS Internals: To the Apple's Core.* Wrox, 2012.

[4] Mark Russinovich, David Solomon, Alex Ionescu *Windows Internals.* Microsoft Press, 2012.

[5] Lukáš Václavík *Android, iOS a nic víc. Podíl mobilní verze Windows je na historickém minimu*
[Online] https://www.cnews.cz/android-ios-nic-vic-podil-mobilni-verze-windows-je-na-historickem-minimu/, 2017.

[6] *802.11a-1999 - IEEE Standard for Telecommunications and Information Exchange Between Systems - LAN/MAN Specific Requirements - Part 11: Wireless Medium Access Control (MAC) and physical layer (PHY) specifications: High Speed Physical Layer in the 5 GHz band.*

[7] *802.11b-1999 - IEEE Standard for Information Technology Telecommunications and information exchange between systems - Local and Metropolitan networks - Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Higher Speed Physical Layer (PHY) Extension in the 2.4 GHz band.*
[Online] http://ieeexplore.ieee.org/document/817038/, 1999.

[8] *802.11g-2003 - IEEE Standard for Information Technology- Telecommunications and Information Exchange Between SystemsLocal and Metropolitan Area Networks- Specific Requirements Part Ii: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.*
[Online] http://ieeexplore.ieee.org/document/1210624/, 2003.

[9] *IEEE Standard for Information technology-Local and metropolitan area networks- Specific requirements-Part 11: Wireless LAN Medium Access Control (MAC)and Physical Layer (PHY) Specifications Amendment 5: Enhancements for Higher Throughput.*
[Online] http://ieeexplore.ieee.org/document/5307322/, 2009.

[10] *802.11ac-2013 - IEEE Standard for Information technology- Telecommunications and information exchange between systemsLocal and metropolitan area networks- Specific requirements-Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications-Amendment 4: Enhancements for Very High Throughput for Operation in Bands below 6 GHz.*
[Online] http://ieeexplore.ieee.org/document/6687187/, 2013.

[11] Gary Scott Malkin *RFC 2453: RIP Version 2.* [Online] https://tools.ietf.org/html/rfc2453, 1998.

[12] J. Moy *RFC 2328: OSPF Version 2.* [Online] https://tools.ietf.org/html/rfc2328, 1998.

[13] Megha Jayakumar, N Ramya Shanthi Rekha, B. Bharathi *A comparative study on RIP and OSPF protocols.* 2015 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS), Coimbatore, 2015, pp. 1-5.

[14] Wei Chen, Chun-mei Liu *The analysis and design of Linux file system based on computer forensic.* 2010 International Conference On Computer Design and Applications, Qinhuangdao, 2010, pp. V2-60-V2-64.

[15] Zheng Wang *Research of data storage mode and recovery method based on XFS file system.* 2016 7th IEEE International Conference on Software Engineering and Service Science (ICSESS), Beijing, 2016, pp. 369-372.

[16] H.T. Reiser *Enhancing ReiserFS security in linux.* Proceedings DARPA Information Survivability Conference and Exposition, 2003, pp. 188 vol.2-.

[17] M. Szeredi *Filesystem in userspace.* [Online] http://fuse.sourceforge.net, 2017.

[18] *RFC 793: Transmission control protocol.* [Online] https://tools.ietf.org/html/rfc793, 1981.

[19] J. Postel *RFC 768: User Datagram Protocol.* [Online] https://tools.ietf.org/html/rfc768, 1980.

[20] Laura A. Chappell *Novell's Guide to LAN/WAN Analysis: IPX/SPXTM.* John Wiley & Sons, 1998.

[21] J. Scott Haugdahl *Inside Netbios.* Architecture Technology Corp, 1990.

[22] A. Milanovic *Performance of UDP and TCP communication on personal computers.* 2000 10th Mediterranean Electrotechnical Conference. Information Technology and Electrotechnology for the Mediterranean Countries. Proceedings. MeleCon 2000 (Cat. No.00CH37099), Lemesos, 2000, vol. 1, pp. 286-289.

[23] W Richard Stevens Bill *Unix Network Programming Volume 1: The S: The Sockets Networking API - Vol. 1.* Dorling Kindesley Pearson Education, 2015.

[24] Alon Atary, Anat Bremler-Barr *Efficient Round-Trip Time monitoring in OpenFlow networks.* IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications, San Francisco, CA, 2016, pp. 1-9.

[25] Xin Liu, Ying Lu, Yutong Lu, Chunjia Wu, Jieting Wu *masFS: File System Based on Memory and SSD in Compute Nodes for High Performance Computers.* 2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS), Wuhan, 2016, pp. 569-576.

[26] Dominik Leibenger, Jonas Fortmann, Christoph Sorge *EncFS goes multi-user: Adding access control to an encrypted file system.* 2016 IEEE Conference on Communications and Network Security (CNS), Philadelphia, PA, 2016, pp. 525-533.

[27] R. Thurlow *RFC 5531: Remote Procedure Call Protocol Specification Version 2.*
[Online] https://tools.ietf.org/html/rfc5531, 2009.

[28] Troy Bryan Downing *Java RMI: Remote Method Invocation.* Wiley, 1998.

[29] Redhat *NFS and portmap.*
[Online] https://access.redhat.com/documentation/enUS/Red Hat Enterprise Linux/5/html/Deployment Guide/s2-nfs-methodology-portmap.html, 2017.

[30] Redhat *Redhat NFS and rpcbind.*
[Online] https://access.redhat.com/documentation/enUS/Red Hat Enterprise Linux/6/html/Storage Administration Guide/s2-nfs-methodology-portmap.html, 2017.

[31] William Gropp, Ewing Lusk, Anthony Skjellum *Using MPI.* MIT Press Cambridge, 1999.

[32] Harry G. Perros *An Introduction to ATM Networks.* Wiley, 2001.

[33] Buyya R., Pathan M., Vakali A. *Content Delivery Networks.* Springer, 2008.

[34] Kenneth P. Birman, Thomas A. Joseph *Reliable communication in the presence of failures.* ACM Transactions on Computer System, 1987, vol. 5, pp. 47-76.

[35] Leslie Lamport *A fast mutual exclusion algorithm.* ACM Transactions on Computer Systems, 1987, vol. 5, pp. 1-11.

[36] Clifford Neuman, Tom Yu, Sam Hartman, Kenneth Raeburn *RFC 4120: The Kerberos Network Authentication Service (V5).*
[Online] https://www.ietf.org/rfc/rfc4120.txt, 2005.

[37] Pavel Bžoch, Jiří Šafařík *Security and reliability of distributed file systems.* Proceedings of the 6th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems, Prague, 2011, pp. 764-769.

[38] Kheng Kok Mar, ZhengQing Hu, Chee Yong Law, Meifen Wang *Secure cloud distributed file system.* 2016 11th International Conference for Internet Technology and Secured Transactions (ICITST), Barcelona, 2016, pp. 176-181.

[39] Madhvaraj M Shetty, D. H. Manjaiah *Data security in Hadoop distributed file system.* 2016 International Conference on Emerging Technological Trends (ICETT), Kollam, 2016, pp. 1-5.

[40] Leslie Lamport *Time, clocks, and the ordering of events in a distributed system.* Communications of the ACM, 1978, vol. 21, pp. 558-565.

[41] Bin Zhou, Yingning Peng, Kris Gaj, Zhonghai Zhou *Implementation and comparative analysis of AES as a stream cipher.* 2009 2nd IEEE International Conference on Computer Science and Information Technology, Beijing, 2009, pp. 396-400.

[42] Ashwak Alabaichi, Faudziah Ahmad, Ramlan Mahmod *Security analysis of blowfish algorithm.* 2013 Second International Conference on Informatics & Applications (ICIA), Lodz, 2013, pp. 12-18.

[43] Zhou Yingbing ; Li Yongzhen *The design and implementation of a symmetric encryption algorithm based on DES.* 2014 IEEE 5th International Conference on Software Engineering and Service Science, Beijing, 2014, pp. 517-520.

[44] Xin Zhou, Xiaofei Tang *Research and implementation of RSA algorithm for encryption and decryption.* Proceedings of 2011 6th International Forum on Strategic Technology, Harbin, Heilongjiang, 2011, pp. 1118-1121.

[45] Eric Rescorla *RFC 2631: Diffie-Hellman Key Agreement Method.* [Online] https://tools.ietf.org/html/rfc2631, 1999.

[46] Christian Bauer *X.509 identity certificates with local verification.* 2012 IEEE International Conference on Communications (ICC), Ottawa, ON, 2012, pp. 6727-6732.

[47] S. Roy, M. Rutherford and C. H. Crawshaw *Towards designing and implementing a secure one time password (OTP) authentication system.* 2016 IEEE 35th International Performance Computing and Communications Conference (IPCCC), Las Vegas, NV, 2016, pp. 1-2.

[48] Jing Bai, Gaoshou Zhai *Study on analysis for SELinux security policy.* 2012 International Conference on Systems and Informatics (ICSAI2012), Yantai, 2012, pp. 1231-1235.

[49] Matějka L. *Distribuované souborové systémy*, Počítačové architektury & diagnostika. Praha: České vysoké učení technické, 2005. pp. 125-128.

[50] Thomas Haynes, David Noveck *RFC 7530: Network File System (NFS) Version 4 Protocol (2015).* [Online] https://tools.ietf.org/html/rfc7530, 2015.

[51] David Mazieres , M. Frans Kaashoek *Self-certifying file system.* Massachusetts Institute of Technology, 2000.

[52] Microsoft *Microsoft DFS Technical Reference.*
[Online] https://technet.microsoft.com/enus/library/cc757042(v=ws.10).
2003.

[53] Microsoft *Microsoft Server Message Block (SMB) Protocol.*
[Online] https://msdn.microsoft.com/en-us/library/cc246231.aspx, 2017.

[54] Christopher R. Hertel *Implementing Cifs: the Common Internet File System.* Prentice Hall Professional Technical Reference, 2003.

[55] R. Eckstein, D. Collier-Brown, P. Kelley *Using Samba.* O'Reilly Associates Inc, Sebastpol, CA, 2000.

[56] M. Satyanarayanan *Coda: a highly available file system for a distributed workstation environment.* Proceedings of the Second Workshop on Workstation Operating Systems, Pacific Grove, CA, 1989, pp. 114-116.

[57] M. Satyanarayanan, Henry H. Mashburn, Puneet Kumar, David C. Steere, James J. Kistler *Lightweight recoverable virtual memory.* Proceedings of the Fourteenth ACM Symposium on Operating Systems Principles, 1993, pp. 146-160.

[58] N. S. Borenstein *CMU's Andrew project: a retrospective.* Communications of the ACM, 1996, vol. 39, no. 298.

[59] Philippas Tsigas *AFS Report Department of Computing Science.* Chalmers Bibliography 90 University of Technology, Goteborg, Sweden, Lecture 2010.

[60] *OpenAFS.*
[Online] http://www.openafs.org, 2017.

[61] *RFC 2663: IP Network Address Translator (NAT) Terminology and Considerations.*
[Online] https://tools.ietf.org/html/rfc2663, 1999.

[62] *TAR.*
[Online] https://www.gnu.org/software/tar/, 2016.

[63] Peter Braam Braam, Michael Callahan *The InterMezzo File System.* Perl Conference, O'Reilly Open Source Convention, Monterey, CA, USA, 1999.

[64] M. O'Connell, P. Nixon *JFS: a secure distributed file system for network computers.* Proceedings 25th EUROMICRO Conference. Informatics: Theory and Practice for the New Millennium, Milan, 1999, vol.2, pp. 450-457.

[65] Edmund B. Nightingale, Jason Flinn *Energy - Efficiency and Storage Flexibility in Blue File System.* Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation, 2004, vol. 6, pp. 25-25.

[66] Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung *The google file system.* Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, 2003, pp. 29-43.

[67] Matějka L., Šafařík J. *Distributed file systems on mobile devices*, Proceedings of the seventh international scientific conference Electronic computers and informatics ECI 2006. Košice: Vienala Press, 2006. pp. 42-47.

[68] Radek Strejc *KIVFS: Souborový a databázový server.* Bakalářská práce, KIV ZČU, 2009.

[69] Jan Fazekaš *KIVFS: Klient pro GNU/Linux s pomocí jaderného modulu.* Bakalářská práce, KIV ZČU, 2010.

[70] Přemysl Jaroš *KIVFS: Klient pro GNU/Linux v FUSE.* Bakalářská práce, KIV ZČU, 2010.

[71] Marek Pivnička *KIVFS: Zabezpečení', šifrování a ověření.* Bakalářská práce, KIV ZČU, 2009.

[72] Radek Strejc *KIVFS - Datové uložiště.* Diplomová práce, KIV ZČU, 2012.

[73] Jindřich Skupa *KIVFS - Synchronizace a trasování požadavků.* Diplomová práce, KIV ZČU, 2012.

[74] Karel Hovorka *KIVFS - Webový klient.* Diplomová práce, KIV ZČU, 2013.

[75] Pavel Bžoch *Zvyšování výkonu a udržování konzistentnosti dat v mobilních zařízeních pro distribuované systémy souborů.* Disertační práce, KIV ZČU 2014.

[76] Pavel Bžoch, Luboš Matějka, Ladislav Pešička, Jiří Šafařík *Towards caching algorithm applicable to mobile clients.* 2012 Federated Conference on Computer Science and Information Systems (FedCSIS), Wroclaw, 2012, pp. 607-614.

[77] Bžoch P., Matějka L., Pešička L., Šafařík J. *Design and Implementation of a Caching Algorithm Applicable to Mobile Clients*, Informatica, 2012, vol. 36, no. 4.

[78] Jindřich Skupa *Dynamické směrování v překryvných sítích s využitím predikce.* Technický report, KIV ZČU, 2015.

[79] Skupa J., Matějka L., Šafařík J. *Dynamic internal message touting in distributed file system.* 2015 IEEE 13th International Scientific Conference on Informatics, Poprad, 2015, pp. 236-240.

[80] Brian Desmond, Joe Richards, Robbie Allen, Alistair G. Lowe-Norris *Active Directory: Designing, Deploying, and Running Active Directory.* O'Reilly Media, 2013.

[81] C. Lonvick *RFC 3164: The BSD Syslog protocol.* [Online] https://www.ietf.org/rfc/rfc3164.txt, 2001.

[82] David V. *Json: Main principals.* CreateSpace Independent Publishing Platform, 2016.

[83] Mark Masse *REST API Design Rulebook.* O'Reilly Media, 2011.

[84] David Gourley, Brian Totty *HTTP: The Definitive Guide (Definitive Guides).* O'Reilly Media, 2002.

[85] Daniel Stenberg *HTTP2 explained.* ACM SIGCOMM Computer Communication Review, 2014, vol. 44, pp. 120-128.

[86] Roy T. Fielding, Julian F. Reschke *RFC 7230: Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing.* [Online] https://tools.ietf.org/html/rfc7230, 2014.

[87] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein *Introduction to Algorithms.* MIT Press, 2009.

[88] Yuval Peres, Dmitry Sotnikov, Benny Sudakov, Uri Zwick *All-pairs shortest paths in $O(n_2)$ time with high probability.* Journal of the ACM, 2013, vol. 60, pp. 1-25.

[89] E. W. Dijkstra *A note on two problems in connexion with graphs.* Numerische Mathematik, 1959, pp. 269-271.

[90] Antun Peicevic *MySQL introduction.* CreateSpace Independent Publishing Platform, 2016.

[91] Ibrar Ahmed, Gregory Smith *PostgreSQL 9.6 High Performance.* Packt Publishing - ebooks Account, 2017.

[92] Stacia Varga, Denny Cherry *Introducing Microsoft SQL Server 2016: Mission-Critical Applications.* Deeper Insights, Hyperscale Cloud. Microsoft Press, 2016.

[93] Bob Bryla, Robert G. Freeman *Oracle Database 12c Release 2 New Features.* McGraw-Hill Education, 2017.

[94] Brent Laster *Professional Git.* Wrox, 2016.

[95] C. Michael Pilato, Ben Collins-Sussman, Brian W. Fitzpatrick *Version Control with Subversion: Next Generation Open Source Version Control.* O'Reilly Media, 2008.

[96] Microsoft *Microsoft Configure an Ini File Item.* [Online] https://technet.microsoft.com/en-us/library/cc731332.aspx, 2017.

[97] G. Carl, G. Kesidis, R.R. Brooks, Suresh Rai *Denial-ofservice attack-detection techniques.* IEEE Internet Computing, vol. 10, no. 1, pp. 82-89, Jan.-Feb. 2006.

[98] K. Narasimha Mallikarjunan, K. Muthupriya, S. Mercy Shalinie *A survey of distributed denial of service attack.* 2016 10th International Conference on Intelligent Systems and Control (ISCO), Coimbatore, 2016, pp. 1-6.

[99] T. Nakashima, S. Oshima, A. Nakashima *Implementation of the performance evaluation system for the NTP server.* 2003 IEEE Pacific Rim Conference on Communications Computers and Signal Processing (PACRIM 2003) (Cat. No.03CH37490), 2003, vol. 2, pp. 828-831.

[100] Rainer Tobbicke *Distributed File Systems: Focus on Andrew File System/Distributed File Service (AFS/DFS).* Proceedings Thirteenth IEEE Symposium on Mass Storage Systems. Toward Distributed Storage and Data Management Systems, Annecy, 1994, pp. 23-26.

[101] John H. Howard, Michael L. Kazar, Sherri G. Menees, David A. Nichols, M. Satyanarayanan, Robert N. Sidebotham, Michael J. West *Scale and performance in a distributed file system.* ACM Transactions on Computer Systems (TOCS), Volume 6 Issue 1, 1988, vol. 6, pp. 51-81.

[102] Matějka L., SKupa J., Strejc R., Pešička L., Šafařík J. *Dynamic Routing in Distributed File System*, Journal of Networks, 2014, vol. 14, pp. 251-259.

[103] Matějka L., Pešička L., Šafařík J. *Distributed file system with online multi-master replicas*, 2011 Second Eastern European Regional Conference on the Engineering of Computer Based Systems, Bratislava, 2011, pp. 13-18.

Seznam obrázků

Seznam tabulek

Seznam použitých zkratek

ACL	Access Control List
AFS	Andrew File System
API	Application Programming Interface
CA	Certification Authority
CDN	Content Delivery Network
CIFS	Common Internet File System
CSD	Circuit Switched Data
DAC	Discretionary Access Control
DB	Database
DFS	Distributed File System
DSS	Distribuovaný souborový systém
DNS	Domain Name Server
DSL	Digital Subscriber Line
ECSD	Enhanced Circuit Switched Data
EDGE	Enhanced Data Rates for GSM Evolution
EGPRS	Enhanced General Packet Radio System
FS	File Server
FTP	File Transfer Protocol
GPRS	General Packet Radio System
HSCSD	High Speed Circuit Switched Data
HSDPA	High Speed Downlink Packet Access
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
I/O	Input/Output
IP	Internet Protocol
ISP	Internet Service Provider
LAN	Local Area Network

LTE	Long Term Evolution
MAC	Mandatory Access Control
NAT	Network Address Translation
NFS	Network File System
NTP	Network Time Protocol
OTP	One time password
RO	Read Only
RPC	Remote Procedure Call
RTT	Round Trip Time
RVM	Recoverable Virtual Memory
RW	Read Write
SD	Secure Digital
SFS	Self-certifying File System
SMB	Server Message Block
SQL	Structured Query Language
REST	Representational State Transfer
TCP	Transmission Control Protocol
TTL	Time to Live
UDP	User Datagram Protocol
UID	User IDentifier
UMTS	Universal Mobile Telephone Standard
USB	Universal Serial Bus
VFS	Virtual File System
VPN	Virtual Private Network
WAN	Wide Area Network
WIFI	Wireless Fidelity
WWW	World Wide Web