diplomová práce

Distribuované úložiště dat - správa metadat

Bc. Tomáš Dyntar

Květen 2014

Ing. Peter Macejko

České vysoké učení technické v Praze
Fakulta elektrotechnická, Katedra počítačů

Poděkování

Děkuji vedoucímu diplomové práce Ing. Peterovi Macejkovi za cenné rady, připomínky a metodické vedení práce. Také bych chtěl poděkovat mé rodině a přátelům za podporu během studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 12. 5. 2014 .....

Abstrakt

Tato práce se zabývá návrhem a implementací subsystému pro správu metadat v systému distribuovaného úložiště dat. Součástí této práce je také přehled cloudových úložišť a výsledky získané jejich testováním.

Klíčová slova

Správa metadata; distribuované úložiště dat; cloudové úložiště

Abstrakt

This diploma thesis is focused on proposal and implementation of the subsystem for metadata management in the system of distributed data storage. The thesis presents also the summary of cloud storages, as well as the results obtained during their testing.

Keywords

Metadata management; distributed data storage; cloud storage

Obsah

Seznam obrázků

Seznam tabulek

Zkratky

AES	Advanced Encryption Standard
API	Application Programming Interface
BSON	Binary JSON
CA	Certificate Authority
CPU	Central Processing Unit
ČVUT	České vysoké učení technické
DDR	Double Data Rate
DHT	Distributed Hash Table
EC2	Elastic Compute Cloud
EXI	Efficient XML Interchange
FEL	Fakulta elektrotechnická
GbE	Gigabit Ethernet
HDD	Hard Disk Drive
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
IP	Internet Protocol
JSON	JavaScript Object Notation
NAS	Network Attached Storage
NAT	Network Address Translation
NIST	National Institute of Standards and Technology
OS	Operační systém
P2P	Peer-to-peer
PDF	Portable Document Format
PGP	Pretty Good Privacy
QR	Quick Response
RAID	Redundant Array of Inexpensive/Independent Disks
RAM	Random-Access Memory
RTF	Rich Text Format
SHA	Secure Hash Algorithm
SSL	Secure Sockets Layer
TLS	Transport Layer Security
VCPU	Virtual Central Processing Unit
W3C	World Wide Web Consortium
XML	Extensible Markup Language

1. Úvod

Cloudové služby a aplikace jsou v poslední době čím dál více populární. Čím dál častěji jsou cloudové aplikace integrovány přímo do operačních systémů počítačů, chytrých telefonů a tabletů. V současné době zažívají největší boom cloudové služby poskytující datové úložiště. Důvodem, proč tomu tak je, je podle odborníků především jejich dostupnost, jednoduchost obsluhy a příkladná synchronizace dat.

Primární funkcí cloudového úložiště je poskytovat úložný prostor. Nicméně většina současných cloudových úložišť poskytuje i spoustu dalších zajímavých rozšíření jako například kancelářské balíky, multimediální přehrávače, mobilní aplikace atd. Cloudová úložiště jednotlivých poskytovatelů se liší svými parametry, tedy spolehlivostí, rychlostí, cenou a zabezpečením dat, a tím pádem jsou vhodná pro různé typy klientů. Cloudové úložiště je ve své podstatě distribuované úložiště dat, protože data jsou distribuována mezi jednotlivými datovými centry poskytovatele této služby.

Tato práce je součástí projektu, ve kterém jsme se spolu s kolegy Bc. Martinem Kudrnáčem a Bc. Janem Janurou nejprve provedli analýzu některých stávajících cloudových úložišť, na základě které jsme vytvořili návrh distribuovaného úložiště dat a na základě tohoto návrhu jsme provedli pilotní implementaci, kterou jsme následně otestovali.

2. Popis problému, specifikace cíle

2.1. Specifikace cíle

Cílem této práce je seznámit se se současnými cloudovými úložišti, podrobit je několika testům a na základě získaných informací navrhnout a implementovat subsystém distribuovaného úložiště dat, jehož funkcí bude správa metadat. Součástí této práce je i návrh zabezpečení distribuovaného úložiště dat a jeho následná implementace. Všechny části distribuovaného úložiště dat jsou navrhovány tak, aby ho bylo možné nasadit ve firemním prostředí, tudíž je kladen důraz na spolehlivost, dostupnost a bezpečnost dat.

Ještě je třeba zmínit, že tato práce je součástí projektu, jehož cílem je vytvoření pilotní implementace distribuovaného úložiště dat. Tento projekt je rozdělen do tří částí. Tato práce se zabývá návrhem subsystému pro správu metadat, práce Martina Kudrnáče [1] se zabývá subsystémem pro správu dat a práce Jana Janury [2] se věnuje klientské části.

2.2. Struktura práce

První část práce se zabývá popisem některých současných cloudových úložišť hlavně z hlediska úložného prostoru, možností sdílení a synchronizace, bezpečnosti a klientských aplikací. Tato úložiště byla následně podrobena několika testům. Výstupem této části je tabulka porovnávající všechna nejznámější současná cloudová úložiště, tedy ta popsaná v této práci a ta, která jsou popsána v pracích Martina Kudrnáče [1] a Jana Janury [2].

Druhá část obsahuje základní koncepci distribuovaného úložiště dat a jsou v ní popsány jednotlivé prvky systému.

Ve třetí části je popsán výběr koncepce a návrh subsystému pro správu metadat. Navíc se tato část zabývá ještě návrhem zabezpečení celého systému.

Čtvrtá část se zabývá realizací subsystému pro správu metadat, která je provedena na základě návrhu popsaného ve třetí části. V této části je popsána implementace jednotlivých funkcí subsystému. Zároveň je v této části popsána implementace zabezpečení celého systému.

Poslední část se věnuje testování. Je v ní popsáno jak testování samotného subsystému pro správu metadat, tak i testování celého systému.

2.3. Základní pojmy

2.3.1. Distribuovaný systém

Distribuovaný systém je počítačový systém složený z více samostatných počítačů, které jsou mezi sebou propojeny sítí a které jsou vybavené softwarem, který počítačům umožňuje koordinovat jejich činnost a sdílet zdroje systémového hardwaru, software a data tak, že se uživateli jeví jako jeden celek [3].

Distribuované úložiště dat je podle definice distribuovaného systému systém poskytující úložný prostor, který se nachází na více počítačích, ale uživateli se jeví jako jeden celek.

2.3.2. Cloud computing

Cloud computing je označení pro metodu přístupu k využití výpočetní techniky, která je založena na poskytování sdílených výpočetních prostředků a jejich využívání formou služby. Existují nejrůznější modely služeb a možnosti jejich poskytování, ale všem typům cloud computingu je společná schopnost poskytovat prostředky na vyžádání, elasticky, samoobslužně a prostřednictvím přístupu z rozsáhlé sítě a také schopnost měřit spotřebované služby v rámci sdíleného fondu prostředků [4].

Cloudové úložiště je služba splňující charakteristiky cloud computingu, která poskytuje úložný prostor.

2.3.3. Metadata

Metadata jsou strukturovaná data, která popisují jiná data. Metadata obsahují základní informace o datech, které zjednodušují práci s konkrétními daty a jejich vyhledávání [5].

3. Přehled cloudových úložišť

V této kapitole jsou popsána některá cloudová úložiště. Tato úložiště byla také podrobena několika testům. Výsledky těchto testů jsou na konci této kapitoly. Popisy a výsledky testů dalších cloudových úložišť jsou v diplomových pracích kolegů Martina Kudrnáče [1] a Jana Janury [2]. V příloze A jsou pak porovnána všechna testovaná cloudová úložiště.

3.1. Google Disk

Google Disk (angl. Google Drive) je cloudové úložiště od společnosti Google, které bylo spuštěno 24. dubna 2012. Umožňuje uživatelům nahrávat, synchronizovat, organizovat a sdílet soubory. Pro použití Google Disku je nutné mít účet Google. Součástí Google Disku je i několik kancelářských aplikací (Dokumenty, Tabulky, Prezentace atd.), které umožňují vytvářet různé druhy dokumentů online, a integrovaný prohlížeč/přehrávač, díky kterému můžete otevřít/přehrát přes 30 typů souborů včetně obrázků, hudby a videa ve vysokém rozlišení [6, 7].

3.1.1. Úložiště

Google nabízí všem uživatelům 15 GB úložného prostoru zdarma, který je společný pro jeho tři nejpoužívanější služby: Google Disk, Gmail a Fotky Google+. Pokud uživateli nestačí 15 GB, může si za měsíční poplatek rozšířit úložný prostor – vybere si jiný plán úložiště (tzv. *storage plan*). Jednotlivé plány úložiště se od sebe liší velikostí úložného prostoru, s čímž souvisí i velikost měsíčního poplatku. V Tab. 1 jsou vypsány všechny plány úložiště, které si lze zaplatit. Důležité je ještě dodat, že pokud si uživatel platí nějaký plán úložiště, potom již nemá k dispozici oněch 15 GB, které jsou zdarma [8, 9].

Velikost úložiště	Cena za měsíc
100 GB	$1,99
1 TB	$9,99
10 TB	$99,99
20 TB	$199,99
30 TB	$299,99

Tabulka 1. Seznam plánů úložiště Google Disk.

Maximální velikost jednoho souboru, který lze na Google Disk uložit, je 10 GB. Toto ovšem neplatí pro soubory vytvořené aplikacemi Dokumenty, Tabulky a Prezentace (kancelářské aplikace, které jsou součástí Google Disku), které mají striktnější omezení. Soubory větší než 25 MB nelze otevřít/přehrát pomocí integrovaného prohlížeče/přehrávače [10].

3.1.2. Použití

Služba Google Disk je svázána s účtem Google, tudíž přihlašovací údaje jsou stejné pro všechny služby poskytované Googlem (např. Gmail, YouTube atd.). S Google Diskem lze pracovat buď přes webové rozhraní, anebo s použitím synchronizačního klienta či mobilní aplikace. Webové rozhraní a všechny ostatní aplikace jsou i v českém jazyce.

Webové rozhraní

Práce s webovým rozhraním je velice jednoduchá a intuitivní. Webové rozhraní se nachází na adrese `https://drive.google.com`. Po přihlášení má uživatel k dispozici veškeré funkce Google Disk. Webové rozhraní poskytuje následující funkce pro práci se soubory:

- vytvoření nové složky,
- nahrání souboru nebo složky,
- stažení souboru nebo složky jako ZIP,
- sdílení souborů a složek,
- přesunutí souboru či složky do jiné složky,
- smazání souboru nebo složky,
- barevně odlišit jednotlivé složky.

Součástí úložiště je i koš. Koš je speciální složka, do které jsou přesouvány veškeré soubory a složky, které uživatel odstranil. Ve webovém rozhraní lze zobrazit obsah koše. Přes webové rozhraní lze koš vysypat, což způsobí fyzické smazání dat. Je nutné mít na paměti, že dokud nejsou data odstraněna z koše, stále uživateli zabírají část úložného prostoru.

Pro některé druhy souborů, přesněji řečeno pro soubory vytvořené aplikacemi z integrovaného kancelářského balíku (tedy Dokumenty, Tabulky a Prezentace), lze nastavit výchozí formát, do kterého se mají před stažením převést. Například soubory vytvořené pomocí aplikace Dokumenty lze před stažením převést do PDF, DOCX, ODT, RTF a HTML.

Aplikace Dokumenty, Tabulky a Prezentace automaticky ukládají a sledují všechny změny, ať už je provede sám uživatel, nebo někdo z jeho spolupracovníků, a to navždy – lze obnovit jakoukoliv předchozí verzi. U ostatních typů souborů je také sledováno, kdo změny provedl, ale ukládají se pouze po 30 dní – lze obnovit maximálně 30 dní starou verzi [11].

Uživatelé používající prohlížeč Google Chrome si mohou nainstalovat webovou aplikaci Google Disk do svého prohlížeče. Tato aplikace jim umožní pracovat s uloženými soubory v offline režimu, tedy ve chvíli, kdy zrovna nemají přístup k internetu. Bohužel v offline režimu lze pracovat pouze se soubory vytvořenými pomocí integrovaného kancelářského balíku [11].

Synchronizační klient

Synchronizační klient Google Disk je k dispozici ve verzi pro operační systém Windows a OS X. Synchronizační klient by měl být integrován i do operačního systému Chrome OS a to od verze 20.

Jelikož synchronizační klient Google Disk umí synchronizovat vše, co se nachází ve stejnojmenné složce, je během instalace uživatel vyzván, aby vybral umístění této složky.

Poté už stačí pouze vyplnit přihlašovací údaje k účtu Google. Od této chvíle se budou veškeré soubory v této složce automaticky synchronizovat.

Synchronizační klient zobrazuje stav synchronizace pouze jako počet přenesených složek a souborů (např. 3 z 6) a umožňuje pozastavit synchronizaci. V nastavení poskytuje synchronizační klient možnost vybrat pouze některé složky ze složky Google Disk, které se budou synchronizovat. Pomocí synchronizačního klienta také nelze nijak pracovat s košem, což způsobuje plýtvání místem v úložném prostoru.

Mobilní aplikace

Mobilní aplikace Google Disk je k dispozici pro operační systémy Android a iOS. Mobilní aplikace svým vzhledem a rozložením komponent se hodně podobá webovému rozhraní a také poskytuje téměř všechny jeho funkce. Oproti webovému rozhraní mobilní aplikace neumožňuje akorát práci s košem. Na druhou stranu součástí mobilního klienta je funkce skenování, díky které si uživatel s chytrým telefonem či tabletem může oskenovat potřebné dokumenty a rovnou je nahrát do Google Disk ve formátu PDF.

3.1.3. Sdílení

Google Disk umožňuje tři druhy sdílení:

- veřejně dostupné na webu,
- všichni uživatelé, kteří mají odkaz,
- sdíleno soukromě.

Zároveň je možné pro každý druh sdílení možné nastavit jednu ze dvou úrovní přístupu:

- může upravovat,
- může prohlížet.

Pokud uživatel zvolí první druh sdílení, tedy veřejně dostupné na webu, mohou se soubory a dokumenty zobrazovat ve výsledcích internetových vyhledávačů a na základě přiřazené úrovně přístupu je může kdokoli prohlížet, nebo dokonce upravovat.

Jestliže uživatel zvolí druhý druh sdílení, tedy všichni uživatelé, kteří mají odkaz, tak k souborům a složkám mohou přistupovat pouze osoby, které mají tento odkaz, a na základě úrovně přístupu je mohou prohlížet nebo upravovat.

Asi nejpoužívanějším druhem sdílení je ten třetí, tedy sdíleno soukromě. V tomto případě může uživatel definovat osoby, které budou mít ke sdílenému souboru či složce přístup, a zároveň pro každou osobu musí nastavit úroveň přístupu.

Velkou výhodou je možnost sdílet soubory, složky a dokonce i soubory aplikací Dokumenty, Tabulky a Prezentace i s osobami, které nemají účet Google.

3.1.4. Bezpečnost

Ze **zásad ochrany soukromí**, se kterými musí uživatel souhlasit při zakládání účtu Google vyplývá, že Google shromažďuje informace o uživateli a o jeho pohybu na internetu, aby mu mohl poskytnout lepší služby – například zobrazit relevantnější výsledky vyhledávání, reklamy atd., a také shromažďuje informace při využívání jejich služeb – například jaké služby uživatel využívá, jakým způsobem apod.

Ze **smluvních podmínek společnosti Google** vyplývá, že Google má možnost kontrolovat obsah jejich služeb, aby zjistil, zda je legální a splňuje zásady společnosti.

Pokud se Google domnívá, že obsah porušuje zásady společnosti nebo právní předpisy, může obsah odstranit nebo zamezit jeho zobrazování. Neznamená to ale, že prověřuje veškerý obsah, pouze ten, u kterého je podezření porušení autorských práv ať už na základě upozornění od jiného uživatele, nebo společnosti vlastnící autorská práva [12, 13].

Google Disk používá pro přenos dat šifrování protokolem SSL. Webové rozhraní a kancelářské aplikace používají pro přenos dat protokol HTTPS, který přenášená data šifruje pomocí SSL, aby uživatele ochránil před neoprávněným přístupem nebo neoprávněným pozměňováním, zveřejněním nebo zničením dat a informací, které uchovává [12].

Google Dive neumožňuje jakkoliv šifrovat obsah. Ale existuje řada aplikací třetích stran, které umožňují jednoduše šifrovat data před uložením na Google Disk. Z těch placených aplikací je to například **Syncdocs** nebo **CloudLock**, z těch aplikací zdarma je to například **Boxcryptor** [14, 15].

Společnost Google nabízí pro přihlášení k účtu Google možnost dvoufázového ověření. Uživatel se k účtu přihlásí standardně pomocí e-mailu a hesla, a navíc je uživatel vyzván k zadání bezpečnostního kódu, který je mu buď zaslán na mobilní telefon pomocí SMS anebo použitím mobilní aplikace Google Authenticator, která přímo v telefonu generuje kódy pro dvoufázové ověření [16].

Bezpečnostní riziko představuje i mobilní aplikace, která k přihlášení využívá účet Google, který je v chytrých telefonech či tabletech přiřazen ve spravovaných účtech, a tudíž není třeba se manuálně přihlašovat. Což znamená, že pokud dojde ke ztrátě nebo odcizení nezabezpečeného chytrého telefonu či tabletu, má kdokoli přístup ke všem souborům uloženým na Google Disk a vlastně i ke všem ostatním službám společnosti Google.

3.1.5. Shrnutí

Cloudové úložiště Google Disk vyniká mezi ostatními úložišti hlavně integrovanými kancelářskými aplikacemi, které poskytují spoustu zajímavých funkcí, zejména pokud uživatel spolupracuje na tvorbě dokumentu, tabulek či prezentace s více uživateli. Největší slabinou Google Disku je bezpečnost, protože používá šifrování pouze pro přenos dat.

3.2. TeamDrive

TeamDrive je cloudová služba německé společnosti TeamDrive Systems GmbH poskytující bezpečnou synchronizaci, úložiště a sdílení souborů jednotlivcům i společnostem [17].

3.2.1. Úložiště

TeamDrive klient je uživateli nabízen ve třech licencích:

- TeamDrive Free,
- TeamDrive Personal,
- TeamDrive Professional.

V rámci všech licencí může uživatel pozvat další uživatele TeamDrive (nezávisle na licenci), aby s ním sdíleli jeho úložný prostor.

Zároveň TeamDrive nabízí uživateli tři alternativy uložení dat:
- TeamDrive cloud,
- vlastní TeamDrive server,
- WebDAV servery.

TeamDrive Free

Jelikož se jedná o licenci zdarma, jsou zde omezení. Velikost úložného prostoru na TeamDrive cloudu může být rozšířena pouze na základě doporučení, tedy pokud uživatel přivede nového uživatele, dostanete 250 MB úložného prostoru navíc. Tímto způsobem může získat až 8 GB, což mu umožní rozšířit úložný prostor na TeamDrive cloudu z 2 GB na 10 GB. Dále pak je tu omezení na straně klienta a to v podobě množství GB, které může TeamDrive klient automaticky synchronizovat, ať už v rámci TeamDrive cloudu, nebo v rámci TeamDrive Personal Serveru či WebDAV serveru. V tomto případě se jedná o 2 GB, jakýkoliv soubor nad tento limit nebude automaticky synchronizován a uživatel musí manuálně potvrdit stažení a nahrání každého souboru [18, 19, 20].

TeamDrive Personal

Tato licence odstraňuje omezení na straně klienta, které se nachází u licence zdarma, a dává uživateli možnost rozšířit si úložný prostor dle svých potřeb. Licence zahrnuje úložný prostor 2 GB na TeamDrive cloudu. Nabízí také vylepšený zákaznický servis a podporu [18, 19, 20].

Zákazník	Cena za rok
Koncový uživatel *(včetně DPH)*	29,99 €
Společnost *(bez DPH)*	25,20 €

Tabulka 2. Cena licence TeamDrive Personal.

TeamDrive Professional

Licence, která obsahuje veškeré funkce licence TeamDrive Personal a nabízí spoustu dalších funkcí, jako například rozšířené možnosti konfigurace, pokročilé verzování, možnost e-mailem zasílat oznámení všem členům týmu apod. Oproti licenci TeamDrive Personal pak může uživatel využít vzdálenou či telefonickou podporu [18, 19, 20].

Zákazník	Cena za rok	Cena za měsíc
Koncový uživatel *(včetně DPH)*	59,99 €	5,99 €
Společnost *(bez DPH)*	50,41 €	5,03 €

Tabulka 3. Cena licence TeamDrive Professional.

TeamDrive cloud

TeamDrive cloud využívá službu Amazon EC2. Při využití TeamDrive cloudu má uživatel ve výchozím nastavení k dispozici 2 GB úložného prostoru zdarma. Velikost úložného prostoru si může uživatel kdykoliv přizpůsobit vlastním potřebám. TeamDrive

cloud uživateli poskytuje maximální ochranu jeho dat, jelikož v TeamDrive cloudu se automaticky vytvoří hned několik šifrovaných bezpečnostních kopií dat [18, 21, 22].

Zvětšení úložného prostoru o	**Cena za měsíc**		**Cena za rok**	
	Koncový uživatel *(včetně DPH)*	Společnost *(bez DPH)*	Koncový uživatel *(včetně DPH)*	Společnost *(bez DPH)*
10 GB	5,99 €	5,03 €	59,99 €	50,41 €
25 GB	14,95 €	12,56 €	149,50 €	125,63 €
50 GB	29,90 €	25,12 €	299,00 €	251,26 €

Tabulka 4. Ceny pro jednotlivá rozšíření úložného prostoru TeamDrive cloudu.

Vlastní TeamDrive server

Společnost TeamDrive Systems GmbH nabízí uživateli možnost synchronizovat svá data s použitím služby TeamDrive na vlastních serverech – má fyzickou kontrolu nad svými daty. TeamDrive server [18, 21, 23] je k dispozici ve dvou variantách:

- **TeamDrive Personal Server** – vhodný pro malé podniky, vzdělávací zařízení a soukromé uživatele. Jedná se o bezpečný HTTP server, který komunikuje výhradně s TeamDrive klientem. Vlastní autentifikační procedury zajišťují, že k serveru mohou přistupovat pouze autorizovaní TeamDrive klienti. Všechna data jsou automaticky ukládána šifrovaná. Je k dispozici pro operační systémy Windows, Linux a OS X [21, 24].
- **TeamDrive Enterprise Server** – vodný pro velké společnosti. Je k dispozici pro operační systém Linux [21].

Služba	**TeamDrive Personal Server Free**		**TeamDrive Personal Server**	
	Koncový uživatel *(včetně DPH)*	Společnost *(bez DPH)*	Koncový uživatel *(včetně DPH)*	Společnost *(bez DPH)*
Cena za rok	Zdarma		99,99 €	84,02 €
Podporovaný úložný prostor[1]	10 GB		Neomezený	

Tabulka 5. Ceny licencí TeamDrive Personal Serveru.

WebDAV servery

TeamDrive umožňuje i synchronizaci dat na WebDAV serverech. WebDAV servery však musejí podporovat celou řadu standardních příkazů, a protože mnoho poskytovatelů WebDAV služeb nastavuje různá omezení, může být obtížné použít tuto technologii pro profesionální účely [21].

3.2.2. Použití

Ke svému účtu TeamDrive se uživatel hlásí pomocí uživatelského jména a hesla, které vyplnil při registraci. S TeamDrive lze pracovat s použitím desktopového klienta nebo

[1] Každý uživatel, který bude chtít s TeamDrive klientem přistupovat k úložnému prostoru většímu než 2 GB, potřebuje minimálně licenci TeamDrive Personal.

mobilní aplikace, které lze přepnout do sedmi různých jazyků. Čeština mezi ně však nepatří.

Desktopový klient

Klientská aplikace TeamDrive je k dispozici ve verzi pro operační systém Windows, OS X a Linux. Po nainstalování klienta uživatel vyplní přihlašovací údaje k účtu TeamDrive. Od této chvíle již může uživatel využívat veškeré funkce TeamDrive. Nejprve je třeba vytvořit novou nebo vybrat existující složku, která má být automaticky synchronizována. Pro každou takto vybranou či vytvořenou složku pak lze vybrat server (viz úložiště), na který se budou data synchronizovat. Takovýchto složek lze vybrat nebo vytvořit nespočetně mnoho. Tyto složky lze samozřejmě sdílet s ostatními osobami, které využívají službu TeamDrive. Synchronizované složky je možné kdykoliv odstranit:

- pouze lokálně,
- pouze na serveru,
- kompletně, tedy lokálně i na serveru.

Pro každou synchronizovanou složku je zároveň vytvořena podsložka koš, do které jsou přesouvány všechny soubory odstraněné z této složky. Soubory v koši lze obnovit do té doby, dokud uživatel koš nevyprázdní. Soubory v koši stále zabírají úložný prostor.

Pro každý soubor, který je změněn ať uživatelem, nebo osobou, se kterou soubor sdílí, se automaticky generuje nová verze souboru. Je tedy možné sledovat, kdo soubor změnil. TeamDrive cloud nijak neomezuje počet uložených verzí. Desktopový klient navíc umožňuje uživateli a spolupracovníkům přidávat komentáře k jednotlivým verzím souborů.

Desktopový klient poskytuje detailní informace o průběhu synchronizace, stavu úložiště a sdílení a také umožňuje vypnout synchronizaci buď pro danou složku, nebo úplně.

Mobilní aplikace

TeamDrive nabízí dvě verze mobilní aplikace:

- Mobilní aplikace TeamDrive, která je zdarma, je k dispozici pro operační systémy Android a iOS. Mobilní aplikace se sice svým vzhledem odlišuje od desktopového klienta, nicméně disponuje stejnými funkcemi a bezpečnostními mechanismy jako desktopový klient.
- Mobilní aplikace TeamDrive SecureOffice, která kombinuje technologii TeamDrive s funkcemi aplikace Picsel Smart Office 2. Tato aplikace je k dispozici pro operační systémy Android a iOS. Jde o klasického mobilního klienta TeamDrive rozšířeného o možnost prohlížet, vytvářet, upravovat a sdílet Microsoft®Office soubory a PDF dokumenty a bezpečně je sdílet a synchronizovat. TeamDrive SecureOffice lze vyzkoušet na 30 dní zdarma, poté je třeba koupit licenci, která stojí 30 €za rok. Bohužel TeamDrive Office licenci je možné využívat jen ve spojení s licencí TeamDrive Professional [25].

3.2.3. Sdílení

TeamDrive umožňuje sdílet pouze synchronizované složky a pouze s osobami, které mají účet TeamDrive. Sdílení je založeno na principu pozvánky. Pro danou složku uži-

vatel vybere seznam osob, se kterými chce složku sdílet. Každé osobě ze seznamu – spolupracovníkovi – lze přiřadit jednu ze čtyř úrovní přístupu:

- **Download Only** – soubory mohou být spolupracovníkem pouze staženy. Změny, které v souborech provede, nebudou synchronizovány.
- **Read/Write** – soubory mohou být spolupracovníkem upravovány.
- **Superuser** – soubory mohou být spolupracovníkem upravovány a zároveň může pozvat ke sdílení další osoby.
- **Administrator** – uživatel/spolupracovník má úplnou kontrolu nad soubory a spolupracovníky a zároveň má možnost soubory nevratně odstranit.

Všem osobám ze seznamu je zaslán e-mail informující pozvánce. Navíc má uživatel možnost zabezpečit pozvánku heslem. Osoba, které přijde pozvánka, má možnost pozvánku přijmout nebo odmítnout. Pokud je pozvánka zabezpečená heslem, je vyzvána k zadání tohoto hesla [26].

TeamDrive nabízí možnost zaslat jednotlivé soubory osobám, které nemají účet TeamDrive anebo TeamDrive účet mají, ale nemají k souborům přístup. V tomto případě je daný soubor zkopírován a nahrán na TeamDrive server v nešifrované podobě. TeamDrive vygeneruje odkaz, odkud je možné daný soubor stáhnout. Tato funkce je dostupná pouze s licencí TeamDrive Professional [26].

3.2.4. Bezpečnost

Společnost TeamDrive Systems GmbH spolupracuje výhradně s bezpečnými certifikovanými datovými centry a je držitelem certifikátu Data Protection Seal of Privacy, který získala na základě auditu Independent Regional Centre for Data Protection of Schleswig-Holstein [27].

Při instalaci TeamDrive klienta je s použitím OpenSSL knihovny vygenerován veřejný a soukromý RSA-2048 klíč. Veřejný klíč je uložen na centrálním registračním serveru TeamDrive [17, 27].

Každá synchronizovaná složka má vygenerovaný svůj vlastní AES-256 klíč, který je uložen pouze na straně klienta. Před tím, než jsou jakákoliv data TeamDrive klientem odeslána, jsou tímto klíčem nejdříve zašifrována. Zároveň tento klíč slouží pro přístup k datům [27, 28].

Jestliže chce vlastník synchronizovanou složku sdílet s jiným uživatelem, zašle mu přes centrální registrační server AES-256 klíč této složky zašifrovaný jeho veřejným RSA-2048 klíčem, který získá z centrálního registračního serveru TeamDrive. Pozvaný uživatel dostane od centrálního registračního serveru zašifrovaný AES-256 klíč, který dešifruje svým soukromým RSA-2048 klíčem. Veškeré soubory ve sdílené složce a změny provedené v této složce jsou zasílány uživatelům, kteří mají k této složce přístup, přes TeamDrive datový server, kde jsou případně uložené v šifrované podobě, pokud zrovna nejsou všichni tito uživatelé online [27, 28].

3.2.5. Shrnutí

Nespornou výhodou cloudové služby TeamDrive je možnost používat službu na vlastních serverech a mít tím pádem fyzickou kontrolu nad svými daty. Tuto možnost ocení zejména společnosti. Naopak velkou nevýhodou oproti ostatním cloudovým úložištím je řada omezení a úložný prostor pouhých 2 GB v licenci zdarma.

3.3. MEGA

Společnost Mega, Ltd. sídlící v Aucklandu na Novém Zélandu spustila cloudové úložiště MEGA 19. ledna 2013 a během první hodiny od spuštění si svůj účet zaregistrovalo na 100000 zájemců. MEGA je následníkem služby Megaupload, která byla 19. ledna 2012 zablokována americkou vládou kvůli obviněním z nelegálního šíření souborů chráněných autorským právem. Stejně jako v případě Megauploadu i za vznikem MEGA stojí internetový podnikatel Kim Dotcom. MEGA umožňuje uživatelům nahrávat, synchronizovat a sdílet soubory [29, 30, 31].

3.3.1. Úložiště

Cloudové úložiště MEGA nabízí bezplatný účet s kapacitou 50 GB. Bezplatný účet není nijak uměle omezen s výjimkou šířky pásma, která je dynamicky nastavována podle vytížení serverů. V rámci bezplatného účtu je k dispozici až šest paralelních slotů pro nahrávání a stahování a také lze omezit rychlost nahrávání [30].

MEGA nabízí tři různé typy profesionálních účtů. V Tab. 6 jsou vypsány parametry jednotlivých profesionálních účtů.

Účet	PRO I	PRO II	PRO III
Velikost úložiště	500 GB	2 TB	4 TB
Šířka pásma	12 TB	48 TB	96 TB
Cena za měsíc	9,99 €	19,99 €	29,99 €
Cena za rok	99,99 €	199,99 €	299,99 €

Tabulka 6. Seznam placených účtů MEGA.

Úložiště MEGA nemá žádné omezení velikosti souboru, takže uživatel je omezen pouze velikostí svého úložiště a technickými limity webového prohlížeče. MEGA disponuje podporou pro přerušené spojení. To znamená, že pokud během nahrávání nebo stahování dojde k výpadku internetu, tak po opětovném připojení MEGA pokračuje v přerušené akci, ovšem za podmínky, že nedojde k zavření prohlížeče [30, 31].

3.3.2. Použití

Uživatel se ke svému účtu hlásí pomocí e-mailové adresy a hesla, které vyplnil při registraci. Heslo musí být dostatečně silné, musí mít dostatečnou délku a obsahovat náhodné znaky, protože slouží jako hlavní šifrovací klíč k účtu. Při prvním přihlášení k účtu MEGA se spustí minihra připomínající legendární hru Arkanoid, která slouží k vygenerování páru RSA-2048 klíčů, které jsou používány pro šifrovací funkce služby. S úložištěm MEGA lze pracovat přes webové rozhraní, anebo s použitím synchronizačního klienta či mobilní aplikace. K dispozici je také aplikace pro prohlížeče Google Chrome a Mozilla Firefox. Webové rozhraní a všechny ostatní aplikace lze přepnout i do českého jazyka [30].

Webové rozhraní

Webové rozhraní se nachází na adrese `https://mega.co.nz` a umožňuje uživateli využívat veškeré funkce úložiště. Disponuje základními funkcemi pro práci se soubory:

- vytvoření nové složky,
- nahrání souboru nebo složky,

- stažení souboru nebo složky (i jako ZIP),
- sdílení souborů a složek,
- přesunutí souboru či složky do jiné složky,
- smazání souboru nebo složky.

Webové rozhraní umožňuje spravovat (přidávat a odebírat) kontakty v seznamu kontaktů. Sdílet soubory a složky lze pouze s uživateli v seznamu kontaktů (pokud nechceme soubor či složku sdílet pomocí odkazu).

Součástí úložiště je i koš. Koš je speciální složka, do které jsou přesouvány veškeré soubory a složky, které uživatel odstranil. Ve webovém rozhraní lze zobrazit obsah koše. Přes webové rozhraní lze koš vysypat, což způsobí fyzické smazání dat. Je nutné mít na paměti, že dokud nejsou data odstraněna z koše, stále uživateli zabírají část úložného prostoru.

Přes webové rozhraní může také uživatel spravovat svůj účet MEGA, pozastavit přenosy souborů a nastavit některé parametry přenosu:

- počet souběžných nahrávání (1-6),
- počet souběžných stahování (1-6),
- omezení rychlosti nahrávání.

Oproti webovým rozhraním ostatních služeb (např. Google Disk) neobsahuje žádný integrovaný prohlížeč obrázků ani přehrávač hudby a videa (viz bezpečnost), zcela zde chybí verzování souborů a seznam posledních akcí uživatele.

Synchronizační klient

Synchronizační klient je zatím k dispozici pouze ve verzi pro operační systém Windows, verze pro operační systémy OS X a Linux by měly být k dispozici začátkem roku 2014 [31]. Po nainstalování klienta si uživatel může vybrat typ synchronizace:

- **Kompletní synchronizace** – v systému vytvořena složka, která je synchronizována s účtem uživatele.
- **Částečná synchronizace** – synchronizovat se budou pouze vybrané složky v účtu. Každé této složce musí uživatel přiřadit složku v systému.

Toto nastavení lze kdykoliv změnit.

Synchronizační klient přehledně zobrazuje průběh synchronizace, seznam nedávno aktualizovaných souborů a využití účtu. Umožňuje nastavit omezení rychlosti nahrávání a pozastavit přenosy souborů. Pomocí synchronizačního klienta nelze nijak pracovat s košem, což způsobuje plýtvání místem v úložném prostoru.

Mobilní aplikace

Mobilní aplikace je k dispozici ve verzi pro operační systémy Android, iOS a BlackBerry OS. Pro operační systém Windows Phone 8 by měla být k dispozici začátkem roku 2014 [30, 32]. Mobilní aplikace svým vzhledem hodně připomíná webové rozhraní a poskytuje téměř všechny funkce webového rozhraní. Oproti webovému rozhraní mobilní aplikace neumožňuje:

- přidat či odebrat kontakt ze seznamu kontaktů,

- sdílet soubor či složku s uživateli ze seznamu kontaktů (lze sdílet pouze pomocí odkazu).

Mobilní aplikace navíc umožňuje nastavit automatickou synchronizaci fotek a videa.

3.3.3. Sdílení

MEGA poskytuje dva druhy sdílení. Prvním způsobem je sdílení pomocí odkazu. Tímto způsobem lze sdílet soubory a složky i s osobami, které nemají účet MEGA. Vygenerovaná adresa má tvar: `https://mega.co.nz/#!AdresaSouboru!TajnýKlíč`. Každý, kdo bude chtít mít přístup k tomuto souboru, potřebuje odkaz na tento soubor a tajný klíč (tajný klíč zde plní roli hesla). Je zde ovšem nebezpečí, že pokud je tajný klíč zveřejněn, může si daný soubor stáhnout kdokoliv, proto tvůrci doporučují nesdílet klíč prostřednictvím nezabezpečených kanálů jako je např. prostý text v e-mailu. Proto je v okně webového rozhraní s vygenerovanou adresou možné odznačit možnost *„Zahrnout klíč souboru“*. Tím pádem se do adresy nezahrne tajný klíč, odkaz je možné bez obav odeslat e-mailem a poté tajný klíč jiným způsobem.

Druhým způsobem je sdílení s uživateli v seznamu kontaktů. Tímto způsobem lze sdílet pouze složky. Ke složce, kterou chce uživatel sdílet, přiřadí daný kontakt ze seznamu kontaktů a pro tento kontakt zvolí jednu ze čtyř úrovní přístupu:

- pouze pro čtení,
- pro čtení i zápis,
- plný přístup,
- odebrat.

Sdílené složce lze přiřadit libovolný počet kontaktů [33].

3.3.4. Bezpečnost

Veškeré šifrování se provádí vždy na koncovém zařízení, tedy nahrávaná data jsou šifrována na zařízení, ze kterého jsou odesílána, a při stahování jsou data dešifrována až po stažení do zařízení. Za generování, výměnu a správu šifrovacích klíčů je tedy zodpovědný klient. Kromě veřejných RSA klíčů žádný z šifrovacích klíčů neopustí koncové zařízení. Ovšem šifrovány nejsou všechny osobní informace. Šifrována jsou pouze data a jména souborů a složek. Informace jako e-mailová adresa, IP adresa, struktura složek, vlastnictví souborů a informace o zaplacení, ke kterým musí úložiště přistupovat, jsou ukládaný a zpracovávány v nešifrované podobě. Protože jsou všechna data v úložišti šifrovaná, nikdo kromě majitele dat neví, co je to za data, a z tohoto důvodu neumí MEGA náhledy obrázků a videí jako je tomu u ostatních úložišť [31, 34].

Na hlavní heslo uživatele do služby MEGA jsou vázány dva klíče – veřejný a soukromý šifrovací klíč. Soukromým klíčem jsou šifrována soukromá data, která uživatel nahrává do svých adresářů, a zároveň po stažení jsou tímto klíčem dešifrována. Veřejný klíč je pak vygenerován pro každý soubor a složku, které bude chtít uživatel sdílet pomocí odkazu. Jelikož hlavní heslo do služby MEGA je i hlavním šifrovacím klíčem ke všem datům, tak pokud ho uživatel ztratí a nemá ke složkám vygenerovaný veřejný klíč pro sdílení, už se k datům nikdy nedostane [31, 34, 35].

Data jsou šifrována pomocí AES-128. Pro sdílení mezi uživateli a ukládání soukromých dat je použit algoritmus RSA-2048. O veškeré šifrování, dešifrování a generování

klíčů se stará JavaScript, který může snížit propustnost na pár MB/s a výrazně vytížit procesor. Tento nedostatek by v budoucnu měla vyřešit technologie HTML5 Web Cryptography API[2] , která bude šifrovat a dešifrovat mnohem rychleji [34].

Data jsou ukládána ve dvou různých datacentrech současně [35].

3.3.5. Shrnutí

Cloudové úložiště MEGA oproti ostatním úložištím vyniká hlavně šifrováním a nabízenou kapacitou 50 GB pro bezplatný účet. Na druhou stranu neobsahuje žádný integrovaný prohlížeč obrázků, přehrávač hudby a videa, kancelářský balík (viz bezpečnost) a zcela zde chybí verzování souborů. Během roku 2014 by mělo být úložiště MEGA rozšířeno o další dvě služby - šifrované zasílání zpráv a šifrovaný video chat [36].

3.4. BitTorrent Sync

BitTorrent Sync je nástroj od společnosti BitTorrent, Inc., který umožňuje sdílet, respektive synchronizovat, soubory mezi zařízeními v lokální síti nebo mezi vzdálenými zařízeními přes Internet s použitím distribuované P2P technologie. Oproti výše zmíněným službám se tedy odlišuje hlavně tím, že zde není žádný server, na který by se data ukládala, ale synchronizace probíhá přímo mezi koncovými zařízeními. Beta verze tohoto nástroje byla do světa vypuštěna 17. července 2013 [37, 38].

3.4.1. Princip

BitTorrent Sync využívá k synchronizaci souborů protokol P2P, který je velice efektivní pro přenos velkých souborů mezi více zařízeními. Data jsou mezi těmito synchronizujícími se zařízeními přenášena po částech a BitTorrent Sync volí takový optimální algoritmus, aby zajistil maximální rychlost stahování a nahrávání [39].

Aby bylo možné najít správná zařízení používající BitTorrent Sync, která mají složku se stejným klíčem, BitTorrent Sync používá:

- **Lokální vyhledávání zařízení** – zařízení v lokální síti vyšle broadcast paket. Pokud se v lokální síti nachází zařízení, které má složku se stejným klíčem, odpoví na broadcast a připojí se.
- **Výměnu informací** – pokud jsou dvě zařízení spojena, vymění si informace o ostatních zařízeních, která znají.
- **Známá zařízení** – v nastavení složky lze přidat zařízení se statickou IP adresou a portem, se kterými se má spojit.
- **DHT** – BitTorrent Sync používá DHT k šíření informací o sobě a k získání informací o jiných zařízeních, které mají složku se stejným klíčem. BitTorrent Sync posílá DHT zprávu ve tvaru: *SHA1(klíč):IP:port*, aby oznámil sám sebe a od DHT obdrží seznam zařízení, která mají stejnou hash *SHA1(klíč)*.
- **BitTorrent tracker** – BitTorent Sync může využívat určitý tracker server, aby usnadnil nalezení zařízení. Na základě kombinace *SHA1(klíč):IP:port* pomáhá tracker server zařízením propojit se přímo. BitTorrent Sync tracker se také chová jako STUN server a pomáhá zařízením navázat přímé spojení skrz NAT.

[2] Web Cryptography API - standard W3C definující API, kterým by prohlížeč měl JavaScriptu zpřístupnit některé kryptografické funkce.

V případě, že zařízení z nějakého důvodu nemůžou komunikovat přímo (např. nacházejí se za silnými firewaly), BitTorrent Sync nabízí možnost komunikace přes relay server [39].

3.4.2. Použití

Jelikož BitTorrent Sync je nástroj pro synchronizaci, a nikoliv služba poskytující synchronizaci, nevyžaduje žádnou registraci. Nástroj BitTorrent Sync je k dispozici jako desktopový klient a mobilní aplikace. Avšak ani jednu z aplikací nelze přepnout do českého jazyka.

Desktopový klient

Klientská aplikace BitTorrent Sync je ve verzi pro operační systém Windows, OS X, Linux a FreeBSD. Po stažení instalátoru, který má jen asi 1,48 MB, nainstalování a nastavení systémových firewallů je BitTorrent Sync připravený k použití. Nyní už stačí jen v aplikaci vybrat složky, které mají být synchronizovány [40].

Uživatelské rozhraní je poněkud strohé, nicméně poskytuje uživateli všechny potřebné funkce. Uživatel zde může nastavovat sdílené a synchronizované složky, sledovat připojená zařízení, sledovat probíhající přenosy, zobrazit historii událostí a samozřejmě má možnost upravit nastavení [40].

Aplikace také podporuje verzování souborů. Starší verze souborů nebo smazané soubory jsou přesouvány do skryté složky *.SyncArchive*, kde jsou ve výchozím nastavení k dispozici ještě 30 dnů, než jsou odstraněny natrvalo. Tato doba se dá v nastavení změnit. Klientská aplikace umožňuje pozastavit synchronizaci a v aplikaci lze také nastavit omezení rychlosti pro stahování a nahrávání [40].

Linuxová verze aplikace trochu zaostává oproti ostatním verzím, jelikož nenabízí klasické desktopové rozhraní, ale jen webové, které navíc neposkytuje všechny funkce desktopového rozhraní. BitTorrent Sync lze také nainstalovat na datová úložiště NAS s operačním systémem Linux a vytvořit si tak vlastní cloudové úložiště [38].

Mobilní aplikace

Mobilní aplikace BitTorrent Sync je ve verzi pro operační systém Android a iOS. Oproti desktopovému klientovi má mobilní aplikace povedené uživatelské rozhraní a disponuje veškerými funkcemi desktopového klienta, a dokonce nabízí i dvě funkce navíc. První takovou funkcí je možnost u sdílené složky zvolit, zda se má synchronizovat automaticky nebo až na vyžádání. Druhou funkcí je možnost jednorázového poslání souborů na mobilní zařízení [38].

3.4.3. Sdílení

Sdílet, respektive synchronizovat, lze pouze složky, nikoliv samostatné soubory. Synchronizace probíhá na základě 32místného tajného klíče označovaného jako secret – tajemství. Pro vygenerování tohoto náhodného tajného klíče využívá BitTorrent Sync soubor /dev/random (OS X, Linux) a Cryptography API (Windows). Pro každou synchronizovanou složku jsou vygenerovány tajné klíče dvou typů:

- pro plný přístup – obsah složky je synchronizován obousměrně,
- pouze pro čtení – obsah složky je synchronizován jednosměrně, tedy obsah složky je synchronizován pouze tam, ale změny se už nesynchronizují zpět.

Zařízení, které se má připojit k synchronizované složce, musí znát tajný klíč. Práva daného zařízení jsou určena typem tajného klíče, který mu byl předán [38].

BitTorrent Sync klientská aplikace navíc umožňuje pro každou synchronizovanou složku vygenerovat jednorázový tajný klíč, který je platný pouze 24 hodin. Jednorázová synchronizace opět může být obousměrná nebo jednosměrná.

Synchronizování složky v počítači se složkou v mobilním zařízení se provádí na základě QR kódu, který je v počítači vygenerován a následně naskenován pomocí kamery mobilního zařízení. QR kód je možné vygenerovat s právem buď pro plný přístup, nebo pouze pro čtení.

K zajištění správné synchronizace složek je nutné, aby kromě synchronizovaného zařízení bylo připojeno ještě alespoň jedno zařízení, které má aktuální data.

3.4.4. Bezpečnost

BitTorrent Sync šifruje data před odesláním pomocí AES-128 klíče v módu CTR. AES-128 klíč je odvozen od klíče synchronizované složky, to znamená, že data může dešifrovat pouze zařízení, které má složku se stejným klíčem [39].

Bezpečnostní riziko může představovat fakt, že BitTorrent Sync nemá integrované žádné vlastní řešení pro předávání klíčů složek, které mají být synchronizovány či sdíleny, takže uživatel může s tímto klíčem libovolně nakládat.

3.4.5. Shrnutí

Nespornou výhodou této technologie je, že vše má pod svou kontrolou uživatel a že velikost úložiště je omezená pouze volným místem na pevném disku zařízení. Největším mínusem této technologie je skutečnost, že správná synchronizace složek je zajištěna jen tehdy, když je připojeno alespoň jedno zařízení, které má aktuální data.

3.5. Testování

Testování cloudových úložišť probíhalo ve dvou fázích. V první fázi probíhalo testování v domácím prostředí, tedy na osobním počítači s domácím připojením k internetu. Ve druhé fázi probíhalo testování na virtuálním stroji připojeném k počítačové síti ČVUT FEL.

Ve všech fázích testování byl pro zaznamenávání datového toku použit program Wireshark.

3.5.1. Testování na osobním počítači

V této fázi testování šlo především o seznámení s výše uvedenými cloudovými úložišti a o získání prvních naměřených hodnot, tj. dobu uploadu, průměrnou rychlost uploadu, dobu downloadu a průměrnou rychlost downloadu. Nicméně tyto hodnoty však nelze porovnávat s hodnotami, které naměřili kolegové u svých úložišť, protože každý z nás měl trochu jinou sestavu pro testování.

Sestava pro testování na osobním počítači:

- Procesor: Intel® Core™ i5-450M @ 2,4 GHz
- RAM: 4,00 GB
- HDD: 500 GB
- OS: Microsoft Windows 7 64bit

- Připojení k internetu:
  - 20 Mb/s download
  - 2 Mb/s upload

V Tab. 7 je seznam testovaných vzorků dat.

Vzorek	Velikost celkem	Obsahuje
Malé soubory	87,8 MB	15761 souborů, 1076 složek
Střední soubory	331 MB	100 souborů, 0 složek
Velký soubor	1495 MB	1 soubor, 0 složek

Tabulka 7. Seznam testovaných vzorků dat na osobním počítači.

Tab. 8 a Tab. 9 zobrazují dobu uploadu (zaokrouhlenou na minuty), průměrnou rychlost uploadu, dobu downloadu (zaokrouhlenou na minuty) a průměrnou dobu downloadu vzorků dat pro jednotlivá úložiště. Ještě je nutné dodat, že měření hodnot u nástroje BitTorrent Sync probíhalo na LAN (100 Mb/s), protože se nepodařilo navázat spojení přes internet.

Na základě údajů v Tab. 8 a Tab. 9 je možné konstatovat, že nejpomalejším a nejméně spolehlivým cloudovým úložištěm je MEGA, které má ve všech měřeních největší dobu uploadu i downloadu, a navíc nedokázalo zpracovat vzorek s malými soubory. Samozřejmě je nutné brát v úvahu to, že výsledky testování mohou být ovlivněny dobou, kdy testování probíhalo.

Upload	Google Disk	TeamDrive	BitTorrent Sync	MEGA
Malé soubory				
Doba uploadu	4h 49m	51m	9m	-
Průměrná rychlost uploadu [Mb/s]	0,04	0,23	1,40	-
Střední soubory				
Doba uploadu	23m	25m	2m	1h 10m
Průměrná rychlost uploadu [Mb/s]	1,92	1,77	26,48	0,63
Velký soubor				
Doba uploadu	1h 46m	1h 48m	2m	>5h[3]
Průměrná rychlost uploadu [Mb/s]	1,98	1,95	105,12	<0,70

Tabulka 8. Měření uploadu na osobním počítači.

3.5.2. Testování na virtuálním stroji

Tato fáze testování byla zaměřena na objektivní porovnání cloudových úložišť, proto testování všech úložišť probíhalo pouze na jednom virtuálním stroji připojeném k počítačové síti ČVUT FEL. Před samotným započetím testování byl vytvořen obraz čistého systému, který byl obnovován pokaždé, když bylo ukončeno testování daného úložiště.

[3] Nahrávání bylo několikrát přerušeno a začínalo od začátku.

Download	Google Disk	TeamDrive	BitTorrent Sync	MEGA
Malé soubory				
Doba downloadu	4h 41m	12m	12m	-
Průměrná rychlost downloadu [Mb/s]	0,04	0,98	1,02	-
Střední soubory				
Doba downloadu	3m	6m	1m	53m
Průměrná rychlost downloadu [Mb/s]	14,71	7,79	55,17	0,83
Velký soubor				
Doba downloadu	11m	43m	2m	15m
Průměrná rychlost downloadu [Mb/s]	19,11	4,89	140,16	14,02

Tabulka 9. Měření downloadu na osobním počítači.

Sestava pro testování na virtuálním stroji:
- Procesor: 2x Intel® Xeon® X5667
- OS: Microsoft Windows 7 64bit
- Připojení k internetu:
  - 1 Gb/s download
  - 1 Gb/s upload

V Tab. 10 je seznam testovaných vzorků dat. Všechny vzorky dat jsou stejné jako v případě testování na osobním počítači až na střední soubory, kde byl počet souborů navýšen na 152.

Vzorek	Velikost celkem	Obsahuje
Malé soubory	87,8 MB	15761 souborů, 1076 složek
Střední soubory	501 MB	152 souborů, 0 složek
Velký soubor	1495 MB	1 soubor, 0 složek

Tabulka 10. Seznam testovaných vzorků dat na virtuálním stroji.

Tab. 11 a Tab. 12 zobrazují dobu uploadu (zaokrouhlenou na minuty), průměrnou rychlost uploadu, dobu downloadu (zaokrouhlenou na minuty) a průměrnou dobu downloadu vzorků dat pro jednotlivá úložiště.

Výsledky testů na virtuálním stroji potvrzují, že ze všech testovaných úložišť je MEGA nejpomalejší a nejméně spolehlivé. Z údajů v Tab. 11 a Tab. 12 je patrné, že ve všech měřeních má největší dobu uploadu i downloadu a navíc nedokázalo vůbec zpracovat vzorek s malými soubory. Samozřejmě je nutné brát v úvahu to, že výsledky testování mohou být ovlivněny dobou, kdy testování probíhalo.

Upload	Google Disk	TeamDrive	BitTorrent Sync	MEGA
Malé soubory				
Doba uploadu	3h 57m	10m	5m	-
Průměrná rychlost uploadu [Mb/s]	0,05	1,17	2,34	-
Střední soubory				
Doba uploadu	14m	8m	3m	51m
Průměrná rychlost uploadu [Mb/s]	4,77	8,60	22,90	1,32
Velký soubor				
Doba uploadu	10m	22m	6m	3m
Průměrná rychlost uploadu [Mb/s]	21,20	9,09	33,50	63,28

Tabulka 11. Měření uploadu na virtuálním stroji.

Download	Google Disk	TeamDrive	BitTorrent Sync	MEGA
Malé soubory				
Doba downloadu	2h 18m	9m	5m	-
Průměrná rychlost downloadu [Mb/s]	0,08	1,31	2,30	-
Střední soubory				
Doba downloadu	3m	4m	3m	11m
Průměrná rychlost downloadu [Mb/s]	26,19	20,35	22,64	6,34
Velký soubor				
Doba downloadu	4m	18m	7m	6m
Průměrná rychlost downloadu [Mb/s]	58,92	10,98	28,61	35,60

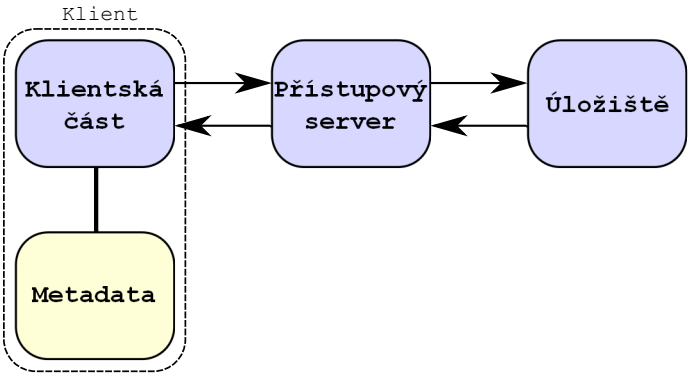
Tabulka 12. Měření downloadu na virtuálním stroji.

4. Koncepce distribuovaného úložiště dat

Tato kapitola popisuje základní koncepci navrhovaného systému distribuovaného úložiště, jednotlivé části systému a jejich funkci v systému. Koncepce systému byla vytvořena ve spolupráci s Martinem Kudrnáčem [1] a Janem Janurou [2].

Návrh systému distribuovaného úložiště je zobrazen na Obr. 1. Celý systém je tvořen třemi částmi:

- *klientská aplikace*,
- *přístupový server* a
- *datové úložiště*.



Obrázek 1. Základní koncepce distribuovaného úložiště dat.

4.1. Klientská aplikace

Klientská aplikace se skládá se tří částí. První část se stará o propojení klientské aplikace s hostitelským souborovým systémem. Druhá část zajišťuje komunikaci s přístupovým serverem a třetí část je zodpovědná za práci s metadaty. Třetí část je realizována pomocí samostatné knihovny. Tato knihovna udržuje informace o souborech uložených v datovém úložišti a o jejich adresářové struktuře.

4.2. Přístupový server

Přístupový server odstiňuje klientskou aplikaci od datového úložiště a zprostředkovává komunikaci mezi nimi. Přístupový server tedy tvoří přístupový bod pro klientské aplikace, které se do systému připojují, a zároveň poskytuje funkce potřebné pro sdílení.

4.3. Datové úložiště

Datové úložiště slouží k samotnému ukládání dat. Toto úložiště tvoří jednotlivá datová centra, která jsou spojena do strukturované sítě pomocí DHT. Každé datové centrum

je navíc schopné obsluhovat požadavky přístupového serveru.

4.4. Komunikace

Komunikace mezi jednotlivými prvky systému je realizována pomocí *socketů*. Pro výměnu informací mezi jednotlivými prvky systému jsou používány informační zprávy, které jsou specifické pro dílčí subsystém a prováděnou operaci.

V rámci celého systému neexistuje žádná databáze uživatelských identifikátorů. Pro identifikaci uživatelů se používá jednoznačný identifikátor, který je vygenerován klientskou aplikací při prvním přihlášení do systému. Tento identifikátor je vytvořen pomocí hashovací funkce SHA-2 a má celkovou délku 80 B.

Pro lokalizaci souborů v rámci DHT se používá prvních 40 B z identifikátoru daného souboru.

4.5. Bezpečnost celého systému

Jednotlivé soubory jsou rozděleny do chunků o maximální velikosti 4 MB, které jsou následně zašifrovány pomocí algoritmu AES-128. Pro každý soubor lze nastavit počet jeho replik (kopií), které budou v datovém úložišti vytvořeny.

Komunikace mezi dílčími prvky systému je zabezpečena pomocí protokolu TLS s PGP mechanismem.

5. Analýza a návrh řešení

Tato kapitola se zabývá analýzou a návrhem subsystému pro správu metadat. Je v ní popsán výběr koncepce subsystému, návrh struktury metadat, výběr formátu chunků metadat a nakonec návrh zabezpečení celého systému.

5.1. Výběr koncepce

Při návrhu subsystému pro správu metadat přicházely v úvahu hned dvě koncepce, ze kterých bylo třeba vybrat tu vhodnější pro naše účely.

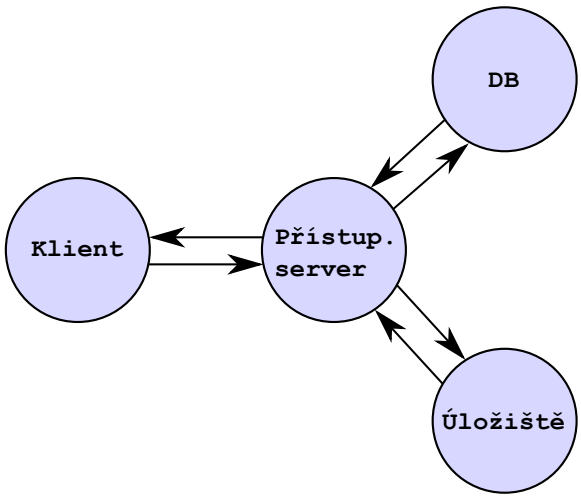
5.1.1. S použitím databáze

První možností bylo pro ukládání metadat využít NoSQL databázi. V systému by z důvodu nespolehlivosti hardware nebyl pouze jeden databázový server, ale bylo by jich hned několik. Ovšem jen jeden by byl primární. Pokud by došlo k výpadku primárního databázového serveru, byl by vybrán jiný databázový server jako primární. Obrovskou výhodou tohoto návrhu je, že o práci s metadaty a o replikaci metadat by se postarala přímo databáze. Tento návrh naopak požaduje chytřejší přístupový server, který bude komunikovat jak s úložištěm, tak s databází.

V úvahu přicházely tyto NoSQL databáze:

- *MongoDB*
- *Redis*
- *CouchDB*

Obr. 2 zobrazuje celkový pohled na systém využívající pro správu metadat databázi.

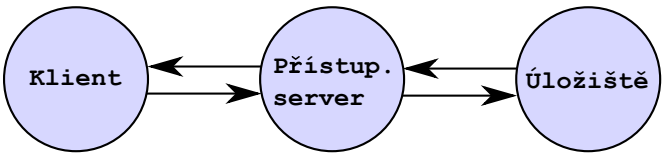


Obrázek 2. Návrh subsystému pro správu metadat využívající databázi.

5.1.2. Bez použití databáze

Druhou možností bylo pro ukládání metadat využít samotné úložiště. V tomto návrhu tedy není zásadní rozdíl mezi metadaty a klasickými daty, protože úložiště se k nim chová stejně. Přesněji řečeno úložiště nepozná rozdíl mezi klasickými daty a metadaty. Přístupový server nebude zatížený jako v prvním případě, protože komunikuje pouze s úložištěm.

Obr. 3 zobrazuje celkový pohled na systém využívající pro správu metadat samotné úložiště.



Obrázek 3. Návrh subsystému pro správu metadat bez použití databáze.

5.1.3. Zvolená koncepce

Jako vhodnější koncepce pro naše účely byla vybrána ta, která pro práci s metadaty nevyužívá databázi. Hlavním důvodem bylo to, že v celkovém návrhu systému odstraníme jeden uzel, tedy databázi, čímž dojde k redukci komunikace v rámci celého systému. Takže funkce, které by plnila databáze v systému, bude nyní plnit úložiště. Mohlo by se zdát, že přesunutím funkce databáze na úložiště nám způsobí komplikovanější návrh úložiště, ale to není tak úplně pravda, protože tytéž funkce, které by databáze poskytovala metadatům, poskytuje i úložiště klasickým datům. Zvolená koncepce nám také ulehčí návrh zabezpečení dat, protože pro ochranu dat a metadat budou použité stejné bezpečnostní mechanismy. Nicméně při použití této koncepce bude třeba mnohem sofistikovanější návrh struktury metadat.

5.2. Návrh struktury metadat

Z hlediska zvolené koncepce budou metadata reprezentována jako klasické soubory, které budou obsahovat všechny potřebné informace o uložených datech. Strukturu metadat bylo potřeba navrhnout tak, aby zajišťovala veškerou potřebnou funkcionalitu.

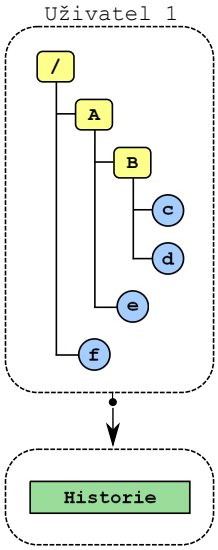
V následujícím textu budu pro *soubor s metadaty* používat pojem **chunk** (česky *kus*) **metadat**.

5.2.1. Chunk metadat

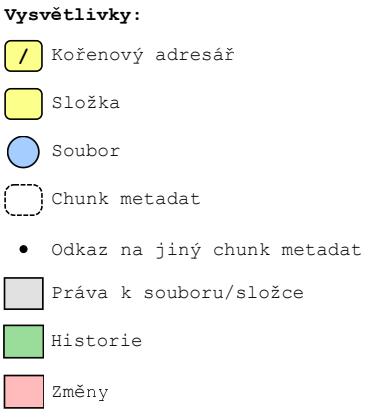
Synchronizace dat mezi klientskou částí a úložištěm probíhá přes namapovaný síťový disk, to znamená, že veškerý obsah tohoto disku je uložen v úložišti. Více v práci Jana Janury o klientské části systému [2].

Protože je třeba zajistit, aby uživatel měl ke svým datům přístup z více klientských aplikací, eventuálně z mobilní aplikace či přes webové rozhraní, musí metadata obsahovat všechny potřebné informace nejen o souborech, ale i o adresářové struktuře uvnitř namapovaného síťového disku.

Obr. 4 zobrazuje návrh chunku metadat. Na Obr. 5 jsou vysvětlivky k tomuto a i k následujícím obrázkům.



Obrázek 4. Ukázka návrhu chunku metadat.



Obrázek 5. Vysvětlivky k obrázkům pro návrh struktury metadat.

Kořenový adresář reprezentuje namapovaný síťový disk, tudíž vše, co se nachází uvnitř tohoto disku, musí být obsaženo v kořenovém adresáři chunku metadat. Aby byla zachována adresářová struktura, je nutné tuto strukturu zahrnout i do metadat. Jediné, co potřebujeme o adresáři vědět je jeho jméno a obsah. U souboru toho potřebujeme vědět o něco více:

- jméno,
- velikost souboru,
- počet replikací souboru,
- seznam chunků daného souboru.

Z důvodu urychlení přenosu souboru do úložiště není soubor přenášen jako celek, ale před samotným odesláním je rozkouskován na tzv. **chunky**, které jsou poté jednotlivě přenášeny do úložiště (více v práci Jana Janury [2]). To znamená, že je potřeba mít v metadatech uloženy odkazy na tyto chunky, aby bylo možné výsledný soubor poskládat zpět.

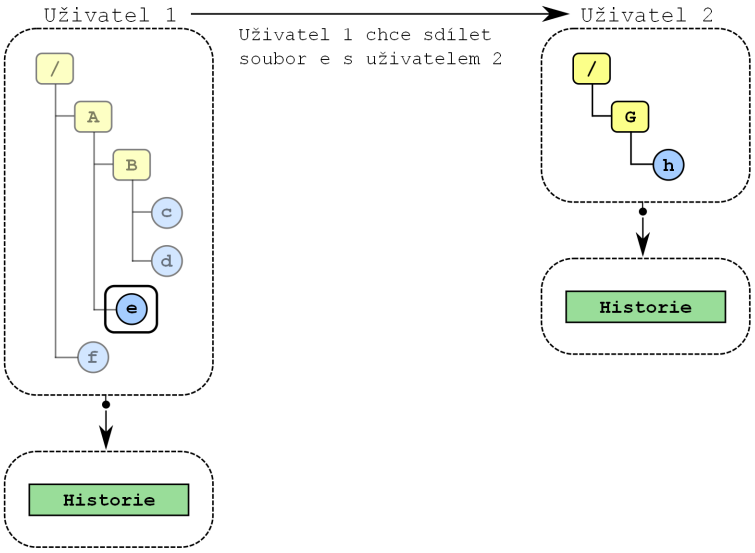
Počet replikací souboru souvisí s jeho důležitostí a znamená to, kolik kopií daného souboru, resp. jeho chunků, bude v úložišti vytvořeno. Více v práci Martina Kudrnáče o správě dat [1]. Tento parametr je tedy také nutné zahrnout do metadat.

Aby bylo možné sledovat změny souborů a případně adresářové struktury, je nutné mít nějakou historii změn. Tato historie je na Obr. 4 reprezentována samostatným chunkem, na který je v hlavním chunku metadat odkaz. Do historie jsou ukládány kompletní informace o souborech a jejich změnách (tzv. *verzování souborů*). To znamená, že uživatel se může kdykoliv vrátit ke starší verzi souboru. S historií adresářové struktury je to trochu problematičtější, protože při každé změně by se do historie musel vložit odkaz na celý chunk metadat, což by vedlo ke zbytečnému zahlcování úložiště. Takže co se týče adresářové struktury, tak do historie budou ukládány pouze informace o změně, tedy např. že došlo k přejmenování složky, k přesunutí složky apod.

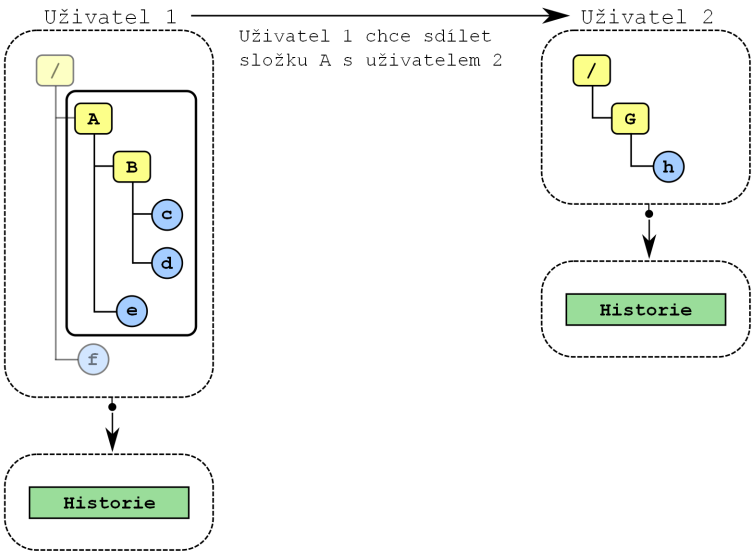
V následujícím textu budu pro tento chunk metadat, tedy chunk metadat obsahující kořenový adresář, používat výraz **uživatelský chunk metadat**, protože kromě daného uživatele nesmí mít nikdo jiný k tomuto chunku metadat přístup.

5.2.2. Sdílení

Aby se nejednalo pouze o systém, kde jsou soubory ukládány a stahovány z úložiště, je nutné tento systém rozšířit o další funkcionalitu. Touto funkcionalitou je sdílení. Koncepcí, jak sdílet data, je mnoho. Můžeme například data mezi uživateli kopírovat, ale tím pádem bude docházet k zahlcování úložiště redundantními daty. Takže nejjednodušším způsobem, jak sdílet data tak, aby nedocházelo ke vzniku zbytečných kopií dat v úložišti, je sdílení na základě metadat.



Obrázek 6. Sdílení - sdílený soubor.



Obrázek 7. Sdílení - sdílená složka.

Na Obr. 6 je zobrazena situace, kdy chce uživatel sdílet pouze jeden soubor s jiným uživatelem. Na Obr. 7 je naopak zobrazena situace, kdy chce uživatel sdílet celou složku s jiným uživatelem. Je těžké určit, jestli je vůbec nutné tyto dva případy rozlišovat, protože principiálně totožné. Nicméně z hlediska budoucího vývoje bylo usouzeno, že

rozlišení, zda je sdílen pouze jeden soubor či celá složka, by mohlo být užitečné.

Na Obr. 8 a Obr. 9 znázorňují návrh sdílení. Jak již bylo řečeno výše, oba návrhy se od sebe principiálně neliší, tudíž v následujícím textu budu popisovat návrh sdílení jak pro sdílený soubor, tak pro sdílenou složku.

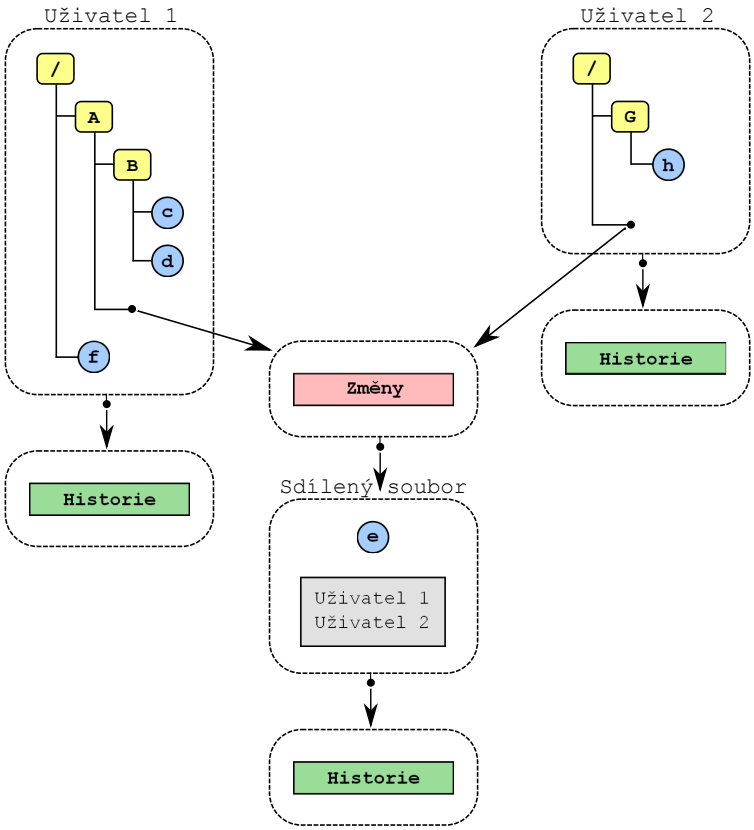
Pokud uživatel bude chtít sdílet nějaký soubor či složku, vytvoří se tři nové chunky metadat:

- změnový chunk,
- sdílený chunk metadat,
- chunk obsahující historii.

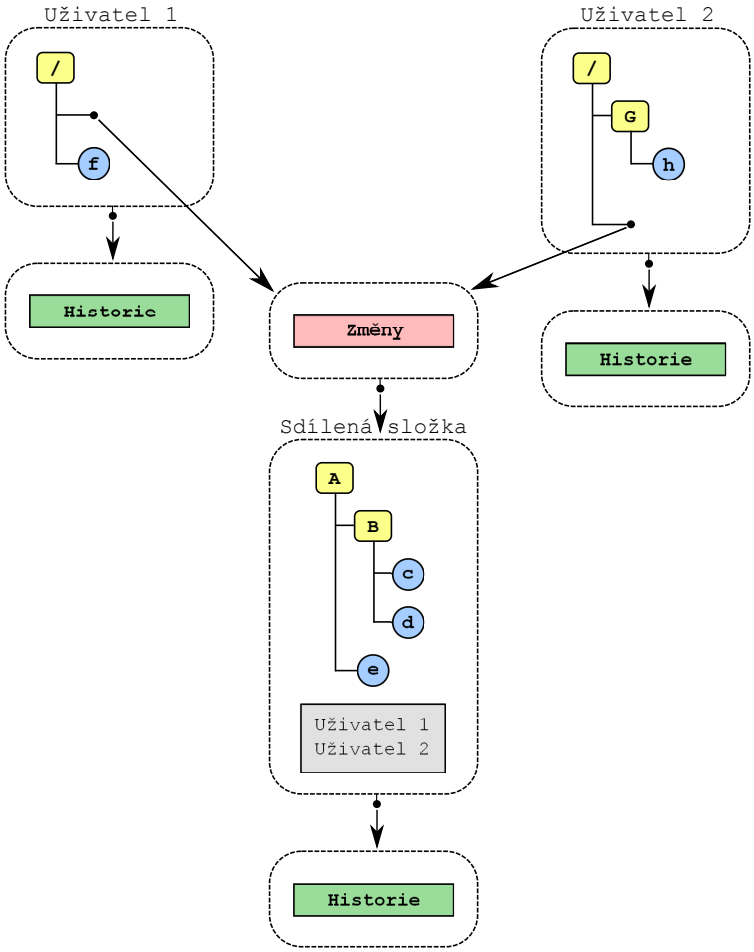
Po sdílení některého souboru či složky jsou veškeré informace o souboru či složce přesunuty z uživatelského chunku metadat do samostatného chunku metadat - sdíleného chunku metadat. V místě uživatelského chunku metadat, kde se původně nacházely informace o souboru či složce, je vytvořen odkaz na změnový chunk, který odkazuje na sdílený chunk metadat. Naopak uživateli, se kterým je daný soubor sdílen, je zaslán odkaz na změnový chunk a tento odkaz je vložen do kořenového adresáře chunku metadat.

Změnový chunk slouží k tomu, aby sdílený soubor či složku mohlo upravovat více uživatelů naráz. Pokaždé, když nějaký uživatel provede změnu ve sdíleném souboru či složce, promítne se tato změna ve změnovém chunku. Tím je zaručeno, že uživatel bude mít vždy přístup k aktuálnímu sdílenému souboru či složce.

Sdílený chunk metadat obsahuje ty samé informace o souborech a složkách jako uživatelský chunk metadat. Navíc jsou však v tomto chunku obsaženy informace o právech



Obrázek 8. Návrh sdílení - sdílený soubor.



Obrázek 9. Návrh sdílení - sdílená složka.

přístupu jednotlivých uživatelů k tomuto souboru či složce. Práva mohou být následující:

- *administrátor* - může číst, upravovat a sdílet s dalšími uživateli,
- *čtení* - může pouze číst,
- *čtení a zápis* - může číst a upravovat.

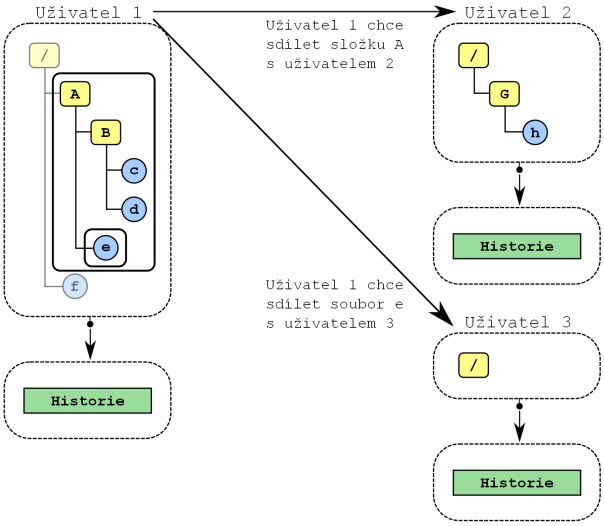
Sdílený chunk metadat, stejně jako uživatelský chunk metadat, obsahuje také odkaz na chunk s historií.

Ukázka složitějšího sdílení je znázorněna na Obr. 10 a Obr. 11.

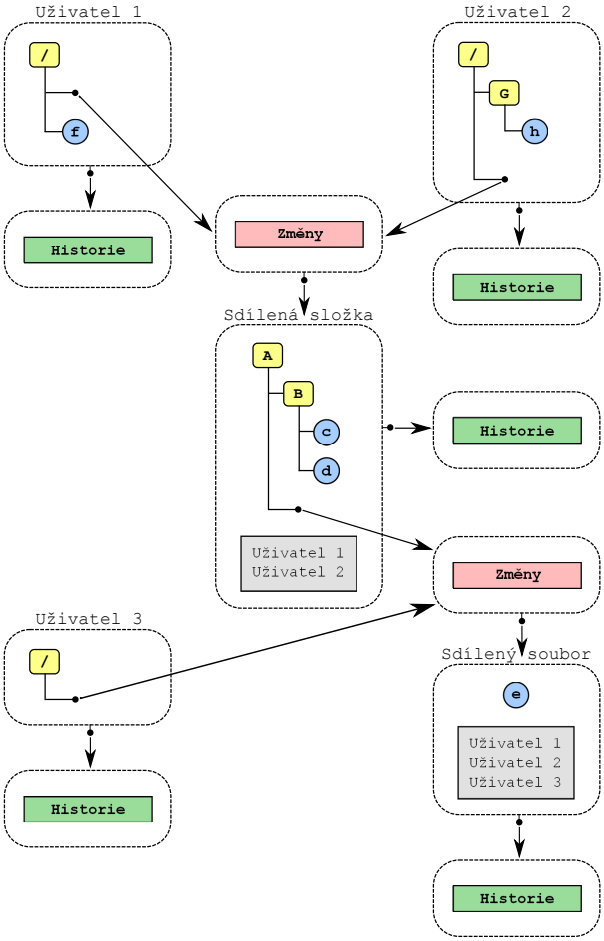
5.2.3. Rozdělení chunku metadat

S rostoucím počtem souborů a složek v namapovaném síťovém disku poroste i velikost chunku s metadaty, což způsobí zpomalení přenosu chunku metadat z úložiště ke klientovi, zejména pokud budeme předpokládat, že uživatel bude k datům přistupovat z mobilní aplikace přes mobilní síť. Aby k této situaci nedocházelo, bylo potřeba vymyslet mechanismus, který by se staral o velikost chunku metadat. Návrh takového mechanizmu je na Obr. 12.

Princip tohoto mechanismu spočívá v tom, že kdykoliv by velikost jakéhokoliv chunku s metadaty překročila maximální povolenou velikost, v metadatech je vyhledána po-



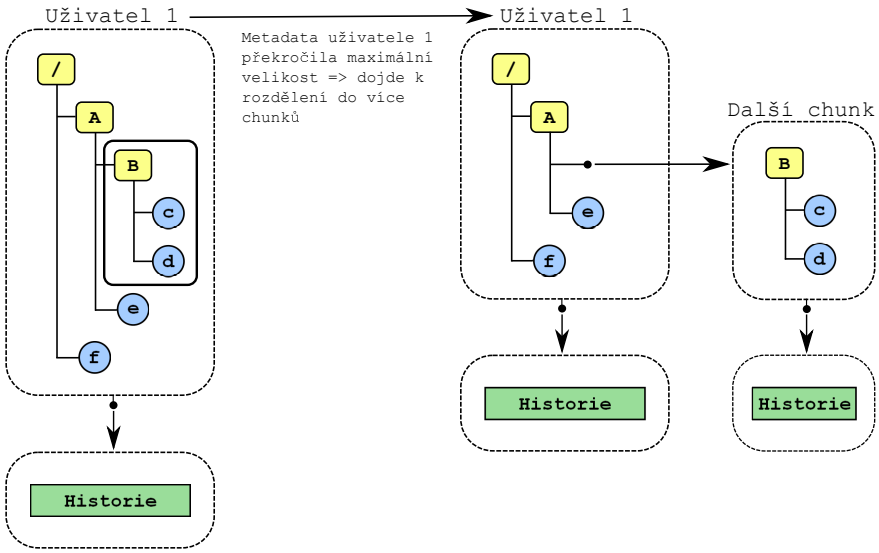
Obrázek 10. Ukázka - složitější požadavek na sdílení.



Obrázek 11. Ukázka - složitější sdílení.

ložka, která zabírá nejvíce místa, a ta je vložena do samostatného chunku metadat. V místě, kde se tato položka původně vyskytovala, je vytvořen odkaz na nově vzniklý chunk metadat. S rozdělením chunku metadat je nutné rozdělit i chunk s historií. Nově vzniklý chunk s historií bude obsahovat pouze záznamy týkající se položek, které obsahuje nově vzniklý chunk metadat.

Načítání chunků metadat bude v klientské aplikaci probíhat tak, že jako první bude načten chunk metadat obsahující kořenový adresář. Další chunky pak budou načítány podle toho, jak bude uživatel pracovat daty.



Obrázek 12. Rozdělení jednoho velkého chunku metadat do více chunků metadat.

5.2.4. Identifikace chunku metadat

Jelikož s chunky metadat je nakládáno stejně jako s klasickými soubory, je třeba, aby byly také stejně identifikované. Víceméně nejjednodušším způsobem, jak jednoznačně identifikovat chunky metadat i klasické soubory, je identifikace na základě hashe neboli otisku. Aby byla minimalizována možnost vzniku kolizních jmen, je nutné zvolit dostatečně silnou hashovací funkci. Na základě článku *Hash Collision Probabilities* [41] a tabulky v článku z Wikipedie: *SHA-2* [42] bylo usouzeno, že pro náš účel je dostačující hashovací funkce **SHA-256**, která vytvoří 256bitovou hash daného chunku metadat. Avšak po hlubší úvaze o tom, kolik souborů může takové úložiště obsahovat, bylo rozhodnuto, že stávající 256bitový výstup SHA-2 bude ještě rozšířen o dalších 64 bitů. Těchto 64 bitů bude získáno tak, že s použitím SHA-256 je vytvořena hash z identifikátoru uživatele a aktuálního času a z 256bitového výstupu je použito pouze prvních 64 bitů.

Takže výsledný identifikátor chunku metadat, resp. jeho jméno, je tvořeno 320bitovou hashí, tedy 80 hexadecimálními znaky.

5.2.5. Přístup k metadatům

Při návrhu mechanismu pro přístup k metadatům bylo potřeba vyřešit dva problémy:

- Jméno chunku metadat je vytvářeno na základě jeho obsahu, identifikátoru uživatele, data a času, takže jakákoliv změna v tomto chunku metadat způsobí změnu

jména tohoto chunku metadat.

- Je třeba zajistit, aby uživatel měl ke svým metadatům přístup z více klientských aplikací.

Oba tyto problémy mohou být vyřešeny speciálním chunkem, který bude mít stálý identifikátor, resp. jméno, které se nikdy nebude měnit. Tento identifikátor musí být jedinečný a zároveň nesmí být poznat, že se jedná o tento chunk.

Proto byl vytvořen chunk obsahující informace o uživateli, který obsahuje mimo doplňujících informací o uživateli také identifikátor uživatelského chunku metadat. Identifikátor, resp. jméno, tohoto chunku tvoří hash privátního klíče uživatele (viz *Návrh zabezpečení systému*). Tímto je zaručena jednoznačnost tohoto jména. Aby nebylo možné odlišit tento chunk od ostatních chunků metadat, musí být jméno tvořeno také 80 hexadecimálními znaky. Jméno tohoto chunku je tedy vytvořeno tak, že hashovací funkce *SHA-512* vytvoří 512bitovou hash privátního klíče. Z této hashe je jako jméno tohoto chunku použito pouze prvních 320 bitů, což nám dá oněch 80 hexadecimálních znaků.

Obrázek 13. Přístup k metadatům.

5.3. Formát chunků metadat

Na základě zvolené koncepce jsou tedy metadata ukládána do souborů, resp. chunků. Bylo tedy nutné rozhodnout, v jakém formátu budou metadata v chunku uložena. V úvahu přicházely dva formáty:

- *XML*,
- *JSON*.

Wikipedie [43] definuje XML jako značkovací jazyk vyvinutý a standardizovaný konsorciem W3C, který je používaný pro serializaci dat a který je určen především pro výměnu dat mezi aplikacemi. JSON definuje Wikipedie [44] jako datový formát nezávislý na počítačové platformě, který je určený pro přenos dat. Jak je patrné z definic, oba formáty se od sebe příliš neliší a z hlediska funkčnosti jsou pro náš účel totožné. Avšak obrovský rozdíl mezi XML a JSON je v počtu a délce tagů, které potřebují k udržení struktury dokumentu. Z tohoto hlediska je XML hodně „upovídané“. Proto na základě vlastních zkušeností a článku *JSON vs. XML: Some hard numbers about verbosity* [45] byl pro uložení metadat zvolen formát JSON. Důvodem byla hlavně veliká „upovídanost“ XML, která by způsobila, že vzniklé chunky s metadaty by byly zbytečně velké.

Samozřejmě daleko vhodnější a z hlediska velikosti chunku metadat úspornější by byly binární verze výše popsaných formátů, tedy BSON a EXI. Ve výsledku by chunky metadat měly být v jednom z těchto formátů, avšak z hlediska vývoje a testování se jeví tyto formáty jako nevhodné.

5.4. Návrh chunků metadat ve formátu JSON

V této části jsou popsány návrhy jednotlivých chunků metadat ve formátu JSON. Při návrhu bylo vycházeno ze standardu ECMA-404: *The JSON Data Interchange Format* [46]. Je třeba ještě dodat, že aby nedocházelo ke zbytečnému navyšování velikosti chunků metadat, byla velikost klíčů v JSON souborech fixně nastavena na čtyři znaky.

5.4.1. Návrh uživatelského chunku matadat

Struktura uživatelského chunku metadat je takováto:

```
{
  "root" : [
    { soubor },
    { složka },
    ...
  ],
  "hist" : string,
  "next" : string
}
```

Hodnota klíče **root** je typu *pole*. Toto pole reprezentuje obsah namapovaného síťového disku a jsou v něm tedy obsaženy objekty reprezentující soubor, složku atd. Struktura těchto objektů je popsána níže. Hodnota klíče **hist** je typu *string* a obsahuje identifikátor chunku s historií, který náleží tomuto chunku metadat. Hodnota klíče **next** je typu *string* a obsahuje identifikátor chunku metadat s pokračováním obsahu kořenového adresáře. Tato položka je tady z toho důvodu, že může nastat situace, kdy v **root** budou pouze odkazy na další chunky metadat (viz níže), ale přitom velikost tohoto chunku metadat bude větší než maximální povolená velikost. V tom případě dojde k roztržení tohoto chunku metadat na dva samostatné chunky metadat, které budou mít menší velikost než je maximální povolená velikost, a v prvním chunku metadat bude do hodnoty klíče **next** vložen identifikátor druhého chunku metadat.

Hodnota typu *string* je prázdná, pokud není klíč v daném případě používán.

5.4.2. Návrh objektu reprezentujícího složku

Objekt reprezentující složku má následující strukturu:

```
{
  "otyp" : 0,
  "name" : string,
  "fils" : [
    { soubor },
    { složka },
    ...
  ]
}
```

Hodnota klíče **otyp** je typu *int* a konkrétně v tomto případě 0. Tato hodnota nám určuje typ objektu. V tomto případě 0 značí, že se jedná o složku. Hodnota klíče **name** je typu *string* a obsahuje jméno složky. Posledním klíčem je **fils**, jehož hodnota je typu *pole*. Toto pole reprezentuje obsah složky, a tudíž obsahuje objekty reprezentující soubor, složku atd.

5.4.3. Návrh objektu reprezentujícího soubor

Objekt reprezentující soubor má následující strukturu:

```
{
  "otyp" : 1,
  "name" : string,
  "size" : string,
  "repl" : int,
  "chns" : [
    { chunk souboru },
    ...
  ]
}
```

Hodnota klíče **otyp** je typu *int* a konkrétně v případě souboru 1. Hodnota klíče **name** je typu *string* a obsahuje jméno souboru. Hodnota klíče **size** je typu *string* a obsahuje celkovou velikost souboru v B. String byl zvolen z toho důvodu, že některé soubory mohou být tak velké, že by došlo k přetečení číselného datového typu. Hodnota klíče **repl** je typu *int*. Toto číslo udává počet replikací, resp. kopií, daného souboru, resp. jeho chunků, v úložišti. Hodnota klíče **chns** je typu *pole*. Toto pole v sobě obsahuje informace o všech chuncích souboru, na které byl soubor rozdělen.

5.4.4. Návrh objektu reprezentujícího chunk souboru

Struktura objektu reprezentujícího chunk souboru je ve následující:

```
{
  "chid" : int,
  "chsz" : int,
  "chli" : string
}
```

Hodnota klíče **chid** je typu *int* a udává pořadové číslo chunku. Hodnota klíče **chsz** udává velikost daného chunku v B a je typu *int*. Tato položka je tu z toho důvodu, že ne všechny chunky souboru musejí mít stejnou velikost. A nakonec hodnota klíče **chli** je typu *string* a obsahuje identifikátor daného chunku.

5.4.5. Návrh objektu odkazujícího na změnový chunk

Jak již bylo zmíněno v části *Návrh struktury metadat*, tak uživatelský chunk metadat obsahuje odkaz na změnový chunk příslušného sdíleného chunku metadat. A také tam bylo řečeno, že chunky metadat, na které je odkazováno z uživatelského chunku metadat, se budou stahovat až tehdy, když je bude uživatel vyžadovat. Proto bylo nutné navrhnout strukturu tohoto objektu tak, aby uživatel měl alespoň nějaké informace o tom, co se ve sdíleném chunku metadat nachází.

Struktura objektu odkazujícího na změnový chunk, který odkazuje na sdílený chunk metadat obsahující sdílenou složku:

```
{
  "otyp" : 2,
  "name" : string,
  "skey" : string,
```

```
  "shar" : string
}
```

Struktura objektu odkazujícího na změnový chunk, který odkazuje na sdílený chunk metadat obsahující sdílený soubor:

```
{
  "otyp" : 3,
  "name" : string,
  "skey" : string,
  "shar" : string
}
```

Oba tyto objekty jsou skoro totožné. Liší se akorát hodnotou klíče **otyp**, jehož hodnota je typu *int* a určuje, jestli uvnitř sdíleného chunku metadat je sdílený soubor či sdílená složka. Hodnota klíče **name** je typu *string* a obsahuje jméno sdíleného souboru či složky. Tato položka je tu zejména proto, aby si každý uživatel mohl sdílený soubor či sdílenou složku pojmenovat podle sebe. Hodnota klíče **skey** je typu *string* a obsahuje klíč, kterým je sdílený chunk metadat zašifrovaný, protože sdílené chunky metadat nejsou šifrovány uživatelským klíčem (více v části *Návrh bezpečnosti*). A nakonec hodnota klíče **shar** je typu *string* a obsahuje identifikátor změnového chunku pro daný sdílený chunk metadat.

5.4.6. Návrh změnového chunku

Struktura změnového chunku není nijak složitá:

```
{
  "updt" : [
    { změna },
    ...
  ]
}
```

Hodnota klíče **updt** je typu *pole*. Toto pole obsahuje objekty reprezentující updaty příslušného sdíleného chunku metadat.

Objekt reprezentující update příslušného sdíleného chunku metadat má strukturu:

```
{
  "time" : string,
  "user" : string,
  "link" : string
}
```

Hodnoty všech klíčů v objektu jsou typu *string*. Hodnota klíče **time** obsahuje datum a čas, kdy byla změna provedena. V hodnotě klíče **user** je obsažen identifikátor uživatele, který změnu provedl. Hodnota posledního klíče, tedy klíče **link**, obsahuje identifikátor příslušného sdíleného chunku metadat.

5.4.7. Návrh sdíleného chunku metadat

Jak již bylo řečeno v části *Návrh struktury metadat*, tak je rozlišován sdílený chunk metadat se sdílenou složkou a sdílený chunk metadat se sdíleným souborem.

Sdílený chunk metadat bude mít pro případ sdílené složky následující strukturu:

```
{
  "shfd" : {
    "otyp" : 0,
    "name" : string,
    "fils" : [
      { soubor },
      { složka },
      ...
    ]
  },
  "shus" : [
    { právo uživatele },
    ...
  ]
  "hist" : string
}
```

To, že se jedná o sdílený chunk metadat je patrné z prvního klíče. Klíč **shfd** nám určuje, že se jedná o sdílený chunk metadat obsahující sdílenou složku. Hodnota tohoto klíče je typu *objekt* a obsahuje onu sdílenou složku. Hodnoty klíčů **otyp**, **name** a **fils** mají stejný význam a typy jako v případě objektu reprezentujícího složku. Hodnota klíče **shus** je typu *pole*. Toto pole obsahuje objekty reprezentující přístupová práva jednotlivých uživatelů. Hodnota klíče **hist** je typu *string* a obsahuje identifikátor chunku s historií, která náleží tomuto sdílenému chunku metadat.

Sdílený chunk metadat bude mít pro případ sdíleného souboru následující strukturu:

```
{
  "shfl" : {
    "otyp" : 1,
    "name" : string,
    "size" : string,
    "repl" : int,
    "chns" : [
      { chunk souboru },
      ...
    ]
  },
  "shus" : [
    { právo uživatele },
    ...
  ]
  "hist" : string
}
```

Struktura sdíleného chunku metadat pro případ sdíleného souboru je téměř totožná se strukturou sdíleného chunku metadat pro případ sdílené složky. Klíč **shfl** nám určuje, že se jedná o sdílený chunk metadat obsahující sdílený soubor. Hodnota tohoto klíče je typu *objekt* a obsahuje onen sdílený soubor. Hodnoty klíčů **otyp**, **name**, **size**, **repl** a **chns** mají stejný význam a typy jako v případě objektu reprezentujícího soubor. Klíče **shus** a **hist** mají stejný význam a typy jako v případě sdíleného chunku metadat obsahujícího sdílenou složku.

5.4.8. Návrh objektu reprezentujícího přístupové právo uživatele

Objekt reprezentující přístupové právo uživatele má následující strukturu:

```
{
  "user" : string,
  "perm" : int
}
```

Hodnota klíče **user** je typu *string* a obsahuje identifikátor uživatele. Hodnota klíče **perm** je typu *int* a obsahuje číslo reprezentující přístupové právo uživatele.

5.4.9. Návrh objektu odkazujícího na další chunk metadat

V části *Návrh struktury metadat* byl popsán mechanismus pro udržování velikosti chunků metadat v nějakých mezích. Když je tato mez překročena, dojde k vyhledání položky, která v uživatelském chunku zabírá nejvíce místa, a ta je přesunuta do samostatného chunku metadat. Na jejím původním místě v uživatelském chunku metadat je vytvořen odkaz na nově vzniklý chunk metadat. Zároveň je tam řečeno, že chunky metadat, na které je odkazováno z uživatelského chunku metadat, se budou stahovat až tehdy, když je bude uživatel vyžadovat. Proto bylo nutné navrhnout strukturu tohoto objektu tak, aby uživatel měl alespoň nějaké informace o tom, co se v dalším chunku metadat nachází.

Struktura objektu odkazujícího na další chunk metadat, který obsahuje složku:

```
{
  "otyp" : 4,
  "name" : string,
  "shar" : string
}
```

Struktura objektu odkazujícího na další chunk metadat, který obsahuje složku:

```
{
  "otyp" : 5,
  "name" : string,
  "shar" : string
}
```

Oba tyto objektu jsou téměř totožné a navíc se moc neliší od objektů, které odkazují na změnový chunk daného sdíleného chunku metadat. Rozdíl je v tom, že v těchto objektech chybí položka s klíčem **skey**. To je z toho důvodu, že k takto nově vzniklým chunkům metadat musí mít přístup pouze uživatel, tudíž jsou šifrovány uživatelským klíčem. Oba tyto objekty se liší pouze v hodnotě klíče **otyp**, která je typu *int*, a která určuje, jestli je v dalším chunku metadat soubor či složka. Hodnoty klíčů **name** a **shar** mají stejný význam a typy jako v případě objektů odkazujících na změnový chunk.

5.4.10. Návrh dalšího chunku metadat

Další chunk metadat obsahující složku má následující strukturu:

```
{
  "nxfd" : {
```

```
    "otyp" : 0,
    "name" : string,
    "fils" : [
      { soubor },
      { složka },
      ...
    ]
  },
  "hist" : string
}
```

Že se jedná o další chunk metadat obsahující složku je patrné z prvního klíče. Klíč **nxfd** nám určuje, že se jedná další chunk metadat obsahující složku. Hodnota tohoto klíče je typu *objekt* a obsahuje danou složku. Hodnoty klíčů **otyp**, **name** a **fils** mají stejný význam a typy jako v případě objektu reprezentujícího složku. Hodnota klíče **hist** je typu *string* a obsahuje identifikátor chunku s historií, která patří tomuto chunku metadat.

Další chunk metadat obsahující soubor má následující strukturu:

```
{
  "nxfl" : {
    "otyp" : 1,
    "name" : string,
    "size" : string,
    "repl" : int,
    "chns" : [
      { chunk souboru },
      ...
    ]
  },
  "hist" : string
}
```

Struktura dalšího chunku metadat obsahujícího soubor je téměř totožná se strukturou dalšího chunku metadat obsahujícího složku. Klíč **nxfl** nám určuje, že se jedná o další chunk metadat obsahující soubor. Hodnota tohoto klíče je typu *objekt* a obsahuje onen soubor. Hodnoty klíčů **otyp**, **name**, **size**, **repl** a **chns** mají stejný význam a typy jako v případě objektu reprezentujícího soubor. Hodnota klíče **hist** je typu *string* a obsahuje identifikátor chunku s historií, která patří tomuto chunku metadat.

5.4.11. Návrh chunku s historií

Struktura chunku s historií je následující:

```
{
  "hist" : [
    {
      "htyp" : 1,
      "path" : string,
      "name" : string,
      "size" : string,
      "repl" : int,
```

```
      "vers" : int,
      "chtm" : string,
      "chus" : string,
      "chns" : [
        { chunk souboru },
        ...
      ]
    },
    {
      "htyp" : 0,
      "oper" : string,
      "chtm" : string,
      "chus" : string
    },
    ...
  ],
  "next" : string
}
```

Chunk obsahující historii začíná klíčem **hist**, jehož hodnotou je *pole*. Toto pole obsahuje všechny změny, které byly provedeny v chunku metadat, ke kterému tento chunk s historií náleží. V tomto poli se mohou nacházet objekty dvou typů. První typ objektu v sobě obsahuje informace o operacích, které byly v příslušném chunku metadat provedeny. Tento typ objektu má hodnotu klíče **htyp** rovnou 0. Druhý typ objektu slouží k verzování souborů a má hodnotu klíče **htyp** rovnou 1.

Objekt obsahující informace o provedených operacích má tedy hodnotu klíče **htyp** rovnou 0. Hodnotou klíče **oper**, která je typu *string*, je popis provedené operace, tedy např. přejmenování, přesunutí, vytvoření, smazání složky či souboru. Hodnoty klíčů **chtm** a **chus** jsou obě typu *string*. Hodnota klíče **chtm** obsahuje datum a čas provedení příslušné operace a hodnota klíče **chus** obsahuje identifikátor uživatele, který danou operaci provedl.

Objekt sloužící k verzování souborů má hodnotu klíče **htyp** rovnou 1. Tento objekt má mnoho totožných položek s objektem reprezentujícím soubor, které mají stejný význam a typ, a proto budou popisovány pouze nové položky. Hodnota klíče **path** je typu *string* a popisuje umístění daného souboru v rámci chunku metadat, ke kterému náleží tento chunk s historií. Hodnota klíče **vers**, která je typu *int*, obsahuje číslo udávající verzi příslušného souboru. Hodnoty klíčů **chtm** a **chus** jsou obě typu *string*. Hodnota klíče **chtm** obsahuje datum a čas vytvoření nové verze a hodnota klíče **chus** obsahuje identifikátor uživatele, který danou verzi vytvořil.

5.4.12. Návrh chunku s informacemi o uživateli

Struktura chunku obsahujícího informace o uživateli je následující:

```
{
  "name" : string,
  "stos" : string,
  "fsts" : string,
  "lcon" : string,
  "root" : string,
  "locs" : string
```

```
}
```

Hodnoty všech klíčů jsou typu *string*. Hodnota klíče **name** obsahuje identifikátor uživatele. Hodnotou klíče **stos** je velikost úložiště v B. Tato položka je tu spíše do budoucna, kdyby se uživatelům přidělovala určitá velikost úložiště. Hodnotou klíče **fsts** je velikost volného místa v úložišti, tedy kolik místa v úložišti uživateli ještě zbývá. Hodnota klíče **lcon** obsahuje pouze informaci, tedy datum a čas, o posledním přístupu k úložišti. Hodnotou klíče **root** je identifikátor uživatelského chunku metadat a hodnota klíče **locs** obsahuje informaci o tom, kam mají být vytvořené chunky metadat ukládány. Z tohoto místa si je potom klientská aplikace vyzvedává a odesílá je do úložiště. Tato položka byla doplněna v průběhu realizace.

5.4.13. Příklad uživatelského chunku metadat

Zde je uveden příklad, jak by mohl vypadat uživatelský chunk metadat:

```
{
  "root" : [
    {
      "otyp" : 0,
      "name" : "Foto",
      "fils" : [
        {
          "otyp" : 1,
          "name" : "Foto1.jpg",
          "size" : "2621440",
          "repl" : 3,
          "chns" : [
            {
            "chid" : 1,
            "chsz" : 1048576,
            "chli" : "1A2B...1A2B"
            },
            {
            "chid" : 2,
            "chsz" : 1048576,
            "chli" : "1B2C...1B2C"
            },
            {
            "chid" : 3,
            "chsz" : 524228,
            "chli" : "1C2D...1C2D"
            }
          ]
        },
        {
          "otyp" : 2,
          "name" : "Dokumenty",
          "skey" : "0123456789ABCDEF0123456789ABCDEF",
          "shar" : "AB23...AB23"
        }
```

```
      ]
    },
    {
      "otyp" : 3,
      "name" : "Instrukce.pdf",
      "skey" : "FEDCBA9876543210FEDCBA9876543210",
      "shar" : "CD45...CD45"
    },
    {
      "otyp" : 0,
      "name" : "Hudba",
      "fils" : [
        {
          "otyp" : 5,
          "name" : "Intro.mp3",
          "next" : "EF67...EF67"
        }
      ]
    },
    {
      "otyp" : 4,
      "name" : "Obrázky",
      "next" : "AA89...AA89"
    }
  ],
  "hist" : "FF99...FF99",
  "next" : ""
}
```


5.5. Návrh zabezpečení systému

Právě bezpečnost dat v cloudových úložištích je v dnešní době často diskutovaným tématem. Každé úložiště používá pro zabezpečení dat trochu jiné mechanismy. Na základě analýzy zabezpečení dat v cloudových úložištích popsaných ve 3. kapitole bylo navrženo zabezpečení celého našeho systému.

Aby se k datům nemohl dostat nikdo, komu nejsou určena, je potřeba zajistit:

- ukládání dat v šifrované podobě,
- bezpečnou komunikaci mezi uzly systému,
- důvěryhodnost komunikujících stran.

Šifrování dat

Data jsou uložená v úložišti v šifrované podobě z toho důvodu, aby nikdo, kdo k tomu nemá oprávnění, nemohl prohlížet uložená dat. Proto je důležité, aby data byla šifrována na straně klienta a aby šifrovací klíč neopustil dané zařízení. Na základě doporučení Národního institutu standardů a technologie (zkr. NIST) popsaných v článku *Cryptographic Key Length Recommendation* [47] bylo usouzeno, že dostatečně silným algoritmem pro šifrování dat bude **AES-128**.

Další bezpečnostní prvek může představovat i to, že jednotlivé soubory nebudou v úložišti uloženy „*tak jak jsou*“, ale před odesláním do úložiště budou rozděleny do více menších částí, které budou následně zašifrovány vybraným algoritmem, tedy AES-128.

Zabezpečení komunikace

Pro zabezpečení komunikace mezi uzly systému byl vybrán protokol **TLS**, který je následníkem protokolu SSL. Oba tyto protokoly fungují na principu asymetrické šifry a poskytují zabezpečení komunikace a autentizaci komunikujících stran. TLS 1.2 (poslední verze, 2008) má oproti SSL 3.0 (poslední verze, 1996) vylepšené některé bezpečnostní mechanismy. TLS navíc oproti SSL umožňuje používání PGP certifikátů, které bude náš systém využívat [48].

Důvěryhodnost

Identifikace komunikujících stran bude v našem systému probíhat na základě digitálních certifikátů. Jelikož se jedná o návrh distribuovaného úložiště dat, nebylo by úplně vhodné použít standard X.509 a zavádět do systému nový prvek v podobě certifikační autority (CA). Z tohoto důvodu se jevilo vhodnější použití standardu **OpenPGP**, který k ověření veřejného klíče nevyužívá CA, ale využívá takzvané sítě důvěry (angl. Web of Trust) [49]. V síti důvěry jsou certifikáty, které obsahují veřejný klíč a informace o vlastníkovi, digitálně podepisované ostatními uživateli sítě důvěry, kteří tím potvrzují propojení tohoto veřejného klíče s osobou nebo subjektem uvedeným v certifikátu. Pro vzájemné podepisování klíčů se organizují tzv. *key signing parties* [50].

6. Realizace

V této kapitole je popsána realizace jednotlivých částí subsystému pro správu metadat. Zároveň tato kapitola obsahuje popis implementace zabezpečení celého systému.

Celá realizace probíhala pod 32bitovým operačním systémem Ubuntu 12.04 LTS. Nicméně celý subsystém pro správu metadat byl implementován s ohledem na přenositelnost na jinou platformu, konkrétně na Windows. Proto byl kladen důraz i na přenositelnost všech použitých knihoven.

6.1. Programovací jazyk

Pro realizaci subsystému pro správu metadat byl zvolen jazyk C++. Důvodem byly hlavně znalosti a zkušenosti s programováním v tomto jazyce, široká dostupnost knihoven a překladačů pro různé platformy, rychlost a výhody objektového programování. Avšak hlavním důvodem bylo, že ostatní části systému byly realizovány v jazyce C.

V částech kódu, které jsou implementovány v jazyce C++, je dodržován standard C++98 a části kódu implementované v jazyce C dodržují standard C90.

6.2. Stručný popis implementace

Z návrhu subsystému pro správu metadat, který byl popsán v předchozí kapitole, vyplývá, že tento subsystém bude třeba propojit s klientskou částí systému. Proto je subsystém pro správu metadat realizován jako *dynamická knihovna*, kterou využívá klientská část systému. Ačkoliv je jádro této knihovny realizováno v jazyce C++, metody a funkce, které knihovna poskytuje, jsou v jazyce C. Takto je to vytvořené z toho důvodu, že klientská část systému, která knihovnu využívá, je implementována v jazyce C.

6.3. Struktura metadat v paměti

Jak již bylo zmíněno v předchozí kapitole, metadata obsahují informace o adresářové struktuře uvnitř namapovaného síťového disku a informace o souborech. Protože adresářová struktura představuje hierarchickou strukturu, která tvoří *strom*, tak nejvhodnějším způsobem, jak reprezentovat metadata v paměti, je pomocí spojové datové struktury, konkrétně N-árního stromu. N-ární strom je takový strom, kde každý uzel může mít 0 až N potomků.

V programu je tato datová struktura realizována třídou `MetaStructure`. Jednotlivé uzly tohoto stromu reprezentuje třída `Node`. Uzel může být tří základní typů:

- ***root***,
- ***složka***,
- ***soubor***.

Jednotlivé typy uzlů jsou rozlišovány na základě proměnné `type` ve třídě `Node`. Uzel typu ***root*** vždy představuje kořen stromu a v metadatech reprezentuje namapovaný síťový disk. ***Root*** obsahuje pouze proměnnou `successors`, která obsahuje pole ukazatelů na další uzly, a proměnnou `history`, která obsahuje identifikátor chunku metadat s historií. Hodnota proměnné `type` je pro uzel typu ***root*** rovna 99.

Uzel typu ***složka*** představuje v metadatech složku a dále se rozděluje do tří podtypů:
- *normální složka*,
- *sdílená složka*,
- *složka v dalším chunku*.

Pokud je uzel typu *normální složka*, je hodnota proměnné `type` rovna 0. Tento uzel představuje v metadatech obyčejnou složku. *Normální složka* obsahuje proměnnou `name`, ve které se nachází jméno složky, a proměnnou `successors`, která obsahuje pole ukazatelů na další uzly.

Sdílená složka je typem uzlu, jehož hodnota proměnné `type` je rovna 2. Tento uzel reprezentuje v metadatech sdílenou složku. *Sdílená složka* obsahuje proměnnou `name`, ve které se nachází jméno této složky. Toto jméno si může uživatel kdykoliv změnit podle svého. Proměnná `originalName` obsahuje původní jméno složky a není možné ho změnit. V proměnné `next` je identifikátor sdíleného chunku metadat, ve kterém je tato složka obsažena. Proměnná `key` obsahuje šifrovací klíč, kterým je daný sdílený chunk metadat zašifrovaný. Proměnná `successors` obsahuje pole ukazatelů na další uzly. Proměnná `share` obsahuje pole práv uživatelů. Na základě tohoto pole je určováno, jak který uživatel může s touto složkou nakládat. Poslední používanou proměnnou je proměnná `history`, která obsahuje identifikátor chunku s historií, který patří danému sdílenému chunku.

Hodnota proměnné `type` pro typ uzlu *složka v dalším chunku* je rovna 4. Tento uzel představuje složku, která je v samostatném chunku metadat. Tento uzel obsahuje stejně jako *normální složka* proměnné `name` a `successors`, které obsahují tytéž hodnoty. Proměnná `next` obsahuje identifikátor chunku metadat, ve kterém se daná složka nachází. Podobně jako *sdílená složka* i *složka v dalším chunku* obsahuje proměnnou `historie`, která obsahuje identifikátor chunku s historií náležícímu danému chunku metadat.

Uzel typu ***soubor*** představuje v metadatech soubor a podobně jako složka se dělí do tří podtypů:
- *normální soubor*,
- *sdílený soubor*,
- *soubor v dalším chunku*.

Jestliže je uzel typu *normální soubor*, je hodnota proměnné `type` rovna 1. Takovýto uzel představuje v metadatech obyčejný soubor. Proměnná `name` obsahuje jméno souboru, `size` jeho velikost v B a `replikacation` počet replikací daného souboru, resp. jeho chunků. Proměnná `chunks` obsahuje pole identifikátorů chunků daného souboru. Objekty v tomto poli jsou instancemi třídy `Chunk`. Tato třída obsahuje proměnné `id`, která obsahuje pořadové číslo chunku, `size`, která obsahuje velikost chunku v B, a `hash`, která obsahuje identifikátor daného chunku daného souboru.

Sdílený soubor je typem uzlu, jehož hodnota proměnné `type` je 3. Tento uzel představuje v metadatech sdílený soubor. Obsahuje ty samé proměnné jako *normální soubor*. Navíc však obsahuje ještě proměnnou `originalName`, která obsahuje původní jméno souboru. Na rozdíl od proměnné `name` nelze toto jméno změnit. V proměnné `next` je

identifikátor sdíleného chunku metadat obsahujícího tento soubor a v proměnné `key` je šifrovací klíč, kterým je daný sdílený chunk metadat zašifrován. Proměnná `share`, stejně jako v případě *sdílené složky*, obsahuje pole práv uživatelů. Poslední proměnnou je `history`, která obsahuje identifikátor chunku s historií patřícímu danému sdílenému chunku metadat.

Hodnota proměnné `type` pro typ uzlu *soubor v dalším chunku* je rovna 5. Tento uzel představuje soubor, který se nachází v samostatném chunku metadat. Tento uzel obsahuje stejné proměnné jako *normální soubor*. Tyto proměnné mají i stejný význam. Navíc obsahuje proměnnou `next`, která obsahuje identifikátor chunku metadat, ve kterém se daný soubor nachází, a proměnnou `history`, která obsahuje identifikátor chunku s historií náležícímu tomuto chunku metadat.

Třída `MetaStructure` poskytuje základní metody a funkce pro práci se *stromem*:

- `getRoot`,
- `addEmptyNode`,
- `addFolderNode`,
- `addFileNode`,
- `deleteNode`,
- `findNode` a
- `findHash`.

Funkce `getRoot` vrací ukazatel na ***root***. Funkce `addEmptyNode` vytvoří na zadaném místě *stromu* nový prázdný uzel. Funkce `addFolderNode` vytvoří na zadaném místě *stromu* a s danými parametry nový uzel představující složku a funkce `addFileNode` vytvoří na zadaném místě *stromu* a s danými parametry nový uzel představující soubor. Metoda `deleteNode` odstraní zadaný uzel ze *stromu*. Funkce `findNode` najde ve *stromu* hledaný uzel. Funkce `findHash` vyhledá ve *stromu* uzel, který obsahuje zadanou hash.

6.4. Práce se soubory ve formátu JSON

Se soubory ve formátu JSON budou prováděny pouze dvě operace - čtení a zápis. V tuto chvíli bylo potřeba rozhodnout se, jestli si napsat vlastní metody pro čtení a zápis ve formátu JSON, anebo použít nějakou existující knihovnu, která poskytuje metody pro práci s JSON soubory. Z časových důvodů byla zvolena druhá možnost, tedy využít nějakou stávající knihovnu. Pro zachování multiplatformnosti celého subsystému je třeba, aby tato knihovna byla také multiplatformní. V úvahu přicházely tyto knihovny:

- *libjson* [51],
- *jzon* [52],
- *JSON Spirit* [53],
- *Jansson* [54] a
- *Rapidjson* [55].

Všechny tyto knihovny jsou k dispozici pod MIT licencí a lze je používat jak pod Linuxem, tak pod Windows, některé i pod OS X. Nakonec byla použita knihovna *jzon*. Důvodem byla hlavně jednoduchost jejího používání a vyhovující dokumentace.

Knihovna je do programu vložena přes hlavičkový soubor `Jzon.h` a je používána ve funkcích a metodách třídy `MetaManipulation`.

6.5. Pomocná knihovna libsfuncs.a

libsfuncs.a je *statická knihovna* realizovaná v jazyce C. Tato knihovna poskytuje funkce, které by jinak bylo nutné realizovat jak v subsystému pro správu metadat, tak v klientské části systému. Knihovna libsfuncs.a využívá některé funkce knihovny *OpenSSL* [56]. Jedná se zejména o funkce implementující SHA-256, SHA-512 a AES-128. libsfuncs.a poskytuje následující metody a funkce:

- genHashSha256_64ext,
- genPrivateKeyHash,
- getCurrentTime,
- encryptFileAES128CTR,
- decryptFileAES128CTR,
- generateAES128Key,
- convertByteToHex,
- convertHexToByte.

Metoda void **genHashSha256_64ext**(char* _output, char* _filePath, char* _username, char* _timestamp) má čtyři parametry:

- _output ... pole velikosti alespoň 81, do kterého metoda uloží vygenerovanou hash,
- _filePath ... umístění souboru, ze kterého se hash bude dělat,
- _username ... uživatelské jméno,
- _timestamp ... datum a čas.

Metoda **genHashSha256_64ext** slouží ke generování identifikátorů souborů a chunků metadat. Tento identifikátor má délku 80 hexadecimálních znaků (320 bitů) a je složen ze dvou částí. První část identifikátoru tvoří SHA-256 hash souboru či chunku metadat a druhou část tvoří prvních 64 bitů z SHA-256 hashe identifikátoru uživatele předaného v parametru _username a data a času předaného v parametru _timestamp.

Metoda void **genPrivateKeyHash**(char* _output, char* _filePath) má dva parametry:

- _output ... pole velikosti alespoň 81, do kterého metoda uloží vygenerovanou hash,
- _filePath ... umístění privátního klíče, ze kterého se hash bude dělat.

Metoda **genPrivateKeyHash** generuje identifikátor z privátního klíče uživatele. Tento identifikátor je používán pro chunk obsahující informace o uživateli, má také délku 80 hexadecimálních znaků (320 bitů) a je vytvářen tak, že je vytvořena SHA-512 hash privátního klíče, ze které je jako identifikátor použito prvních 320 bitů.

Metoda void **getCurrentTime**(char* _output) má jeden parametr:

- _output ... pole velikosti alespoň 20, do kterého metoda uloží aktuální čas.

Metoda **getCurrentTime** je používána pro získání aktuálního času operačního systému. Čas má formát „dd.mm.YYYY HH:MM:SS“.

Funkce int **encryptFileAES128CTR**(const char* _readFilePath, const char* _writeFilePath, const char* _hexKey) má tři parametry:

- _readFilePath ... umístění souboru, jehož obsah se bude šifrovat,
- _writeFilePath ... umístění souboru, do kterého se bude zapisovat zašifrovaný obsah (pokud tento soubor nebude existovat, vytvoří se),

- `_hexKey` ... šifrovací klíč, který musí mít délku 32 hexadecimálních znaků (128 bitů).

Funkce **`encryptFileAES128CTR`** slouží k šifrování souborů a chunků metadat. Pro šifrování používá algoritmus AES-128 v módu CTR. Tato funkce vrací 0, pokud vše proběhne v pořádku, jinak číslo menší než 0.

Funkce `int` **`decryptFileAES128CTR`**`(const char* _readFilePath, const char* _writeFilePath, const char* _hexKey)` má tři parametry:

- `_readFilePath` ... umístění souboru, jehož obsah se bude dešifrovat,
- `_writeFilePath` ... umístění souboru, do kterého se bude zapisovat dešifrovaný obsah (pokud tento soubor nebude existovat, vytvoří se),
- `_hexKey` ... dešifrovací klíč, který musí mít délku 32 hexadecimálních znaků (128 bitů).

Funkce **`decryptFileAES128CTR`** slouží k dešifrování souborů a chunků metadat, které byly zašifrovány metodou `encryptFileAES128CTR`. Tato funkce vrací 0, pokud vše proběhne v pořádku, jinak číslo menší než 0.

Metoda `void` **`generateAES128Key`**`(unsigned char* _key)` má jeden parametr:

- `_key` ... pole velikosti alespoň 16, do kterého bude uložen vygenerovaný klíč.

Metoda **`generateAES128Key`** je používána pro náhodné generování 128bitového (16bytového) šifrovacího klíče. Tato metoda využívá ke generování náhodných bytů funkci `RAND_bytes` implementovanou v knihovně *OpenSSL* [56].

Metoda `void` **`convertHexToByte`**`(unsigned char* _bytes, const char* _hex)` má dva parametry:

- `_bytes` ... pole velikostí alespoň 16, do kterého bude uložen šifrovací klíč v bytové podobě,
- `_hex` ... pole, ve kterém je šifrovací klíč v hexadecimální podobě.

Metoda `void` **`convertByteToHex`**`(char* _hex, unsigned char* _bytes)` má dva parametry:

- `_hex` ... pole velikosti alespoň 33, do kterého bude uložen šifrovací klíč v hexadecimální podobě,
- `_bytes` ... pole, ve kterém je šifrovací klíč uložen v bytové podobě.

Metody **`convertByteToHex`** a **`convertHexToByte`** slouží k převodu 128bitového šifrovacího klíče z bytové podoby do hexadecimální podoby a naopak.

6.6. Knihovna `libMetaManagement.so`

Knihovna `libMetaManagement.so` je výsledkem realizace subsystému pro správu metadat. Jedná se o *dynamickou knihovnu* poskytující veškeré potřebné funkce a metody pro práci s metadaty.

6.6.1. Použité struktury

Pro výměnu dat mezi knihovnou a klientskou aplikací jsou v některých případech potřeba složitější datové struktury. Některé funkce této knihovny používají následující struktury:

- CChunk,
- CChunkArray,
- FolderContent,
- FolderContentArray,
- HashChunk,
- HashArray,
- UserPerm a
- UserPermArray.

```
typedef struct {
  int id;
  unsigned size;
  const char* hash;
} CChunk;
```

Struktura **CChunk** představuje jeden chunk souboru. Tato struktura obsahuje tři položky. Položka id obsahuje pořadové číslo chunku, v položce size je velikost chunku a položka hash obsahuje identifikátor, tedy hash, chunku.

```
typedef struct {
  int arraySize;
  CChunk* cchunkArray;
} CChunkArray;
```

Struktura **CChunkArray** slouží pro přenos více chunků souboru, resp. více struktur CChunk. Tyto struktury jsou uložené v poli cchunkArray. Velikost tohoto pole je obsažena v položce arraySize.

```
typedef struct {
  int type;
  uint64_t size;
  const char* name;
} FolderContent;
```

FolderContent je struktura představující jeden prvek adresářové struktury metadat. Hodnota položky type určuje, zda je jedná o soubor či složku. Pro případ souboru určuje položka size jeho velikost, naopak pro případ složky obsahuje tato položka 0. Položka name obsahuje jméno daného souboru nebo složky.

```
typedef struct {
  int arraySize;
  FolderContent* fContentArray;
} FolderContentArray;
```

Struktura **FolderContentArray** slouží pro přenos více prvků adresářové struktury metadat najednou, tedy struktur FolderContent, které jsou uložené v poli fContentArray. Položka arraySize obsahuje velikost tohoto pole.

```
typedef struct {
  int type;
  uint64_t size;
  char* key;
  char* hash;
} HashChunk;
```

HashChunk je struktura, která v různých situacích slouží pro přenos různých informací. Primárně je tato struktura určena pro přenos informace o vytvořeném chunku metadat. V tomto případě hodnota položky `type` určuje typ daného chunku. Tato hodnota je rovna 0, pokud se jedná o chunk metadat, nebo 1, pokud se jedná o chunk s uživatelskými informacemi, anebo 2, pokud se jedná o sdílený chunk metadat. Položka `hash` obsahuje identifikátor, resp. hash, daného chunku. Pokud je hodnota položky `type` rovna 2, je v položce `key` obsažen šifrovací klíč, kterým je daný chunk zašifrován. V ostatních případech je tato položka prázdná. Položka `size` není v tomto případě používána a obsahuje 0. Tato struktura však slouží i pro přenos informace o velikosti souboru nebo o počtu replikací daného souboru. V obou případech obsahuje položka `type` 0 a v položce `size` je buď velikost souboru, nebo počet replikací souboru. Ostatní položky jsou prázdné.

```
typedef struct {
  int arraySize;
  HashChunk* hashArray;
} HashArray;
```

Struktura **HashArray** je určená pro přenos informací o vytvořených chuncích metadat. Informace o jednotlivých chuncích metadat jsou obsažené v poli `hashArray`. Velikost tohoto pole je obsažena v položce `arraySize`.

```
typedef struct {
  const char* user;
  int perm;
} UserPerm;
```

UserPerm je struktura představující přístupové právo uživatele k danému sdílenému souboru či sdílené složce. Tato struktura obsahuje pouze dvě položky. Položka `user` obsahuje identifikátor uživatele a položka `perm` určuje typ přístupového práva.

```
typedef struct {
  int arraySize;
  UserPerm* userPermArray;
} UserPermArray;
```

Struktura **UserPermArray** slouží pro přenos více struktur `UserPerm`, tedy více uživatelských práv. Struktury `UserPerm` jsou uložené v poli `userPermArray`, jehož velikost je obsažena v položce `arraySize`.

6.6.2. Inicializace a ukončení

Dříve než je možné používat funkce a metody poskytované touto knihovnou, je třeba inicializovat všechny vnitřní objekty. K tomu slouží metoda `void meta_Init(const char* _key, const char* _localChunkStoragePath, const char* _userChunkHash)`, která má tři parametry:

- _key ... uživatelský šifrovací klíč,
- _localChunkStoragePath ... místo, kam jsou ukládány vytvořené chunky metadat,
- _userChunkHash ... hash chunku s uživatelskými informacemi.

V parametrech metody **meta_Init** jsou předávány informace, které potřebuje téměř každá funkce knihovny. Tato metoda je uloží do vnitřních proměnných knihovny. Parametr _key předává uživatelský šifrovací klíč, kterým jsou šifrovány všechny chunky metadat, mimo sdílených chunků metadat samozřejmě. Parametr _localChunkStoragePath předává místo, kam jsou ukládány vytvořené chunky metadat a odkud si je následně vyzvedává klientská aplikace a odesílá je do úložiště. Zároveň jsou z tohoto místa chunky metadat načítány. V parametru _userChunkHash je předávána hash chunku s uživatelskými informacemi, která je vytvořena z privátního klíče uživatele.

K uvolnění veškerých alokovaných prostředků slouží metoda void **meta_Exit()**, která by měla být volána před samotným ukončením klientské aplikace.

Do této skupiny patří ještě funkce HashArray **meta_firstLaunch**(const char* _username, uint64_t _storageSize, const char* _lastConnect), která musí být volána pouze při prvním spuštění klientské aplikace. Tato metoda vytvoří první dva chunky metadat - chunk s uživatelskými informacemi a prázdný uživatelský chunk metadat. Tato funkce vrací strukturu HashArray, která obsahuje informace vytvořených chuncích. Funkce **meta_firstLaunch** má tři parametry:

- _username ... identifikátor uživatele,
- _storageSize ... velikost úložiště,
- _lastConnect ... datum a čas posledního přihlášení uživatele.

V těchto parametrech jsou předávány informace, které jsou vloženy do chunku s uživatelskými informacemi. V parametru _username je předáván identifikátor uživatele. Tento identifikátor se používá při sdílení. Parametr _storageSize předává informaci o velikosti místa v úložišti, které má uživatel přidělené. Tato hodnota se zatím nepoužívá, ale mohla by být využita v budoucnu. Datum a čas poslední provedené změny metadat jsou předávány v parametru _lastConnect. Tento parametr je vkládán do každého nově vytvářeného chunku s uživatelskými informacemi a má pouze informativní charakter.

6.6.3. Funkce pracující s informacemi o uživateli

Jak již bylo mnohokrát zmíněno, chunk s informacemi o uživateli představuje jakýsi přístupový bod k metadatům. Tento chunk je jediný, u kterého se identifikátor, resp. hash, nemění. Chunk s uživatelskými informacemi je vytvořen ve funkci **meta_firstLaunch**. Pro načtení informací z tohoto chunku slouží funkce int **meta_user_loadChunk**(const char* _fileName). Tato metoda má jeden parametr:

- _fileName ... identifikátor chunku s uživatelskými informacemi.

Tato funkce si v místě, které bylo předáno v metodě **meta_Init** parametrem _localChunkStoragePath, vyhledá chunk s daným identifikátorem předaným ve _fileName. Tento chunk následně dešifruje a informace z něj načte do vnitřního objektu metaUser. Tento objekt je instancí třídy MetaUser. Tato funkce vrací 0, pokud vše proběhne v pořádku, jinak číslo menší než 0.

Dalšími funkcemi jsou jen tzv. *gettery* a *settery*. Gettery slouží k získání hodnot jednotlivých proměnných objektu `metaUser`. Mezi gettery patří funkce:

- `const char* meta_user_getName()`,
- `uint64_t meta_user_getStorageSize()`,
- `uint64_t meta_user_getFreeStorageSize()`,
- `const char* meta_user_getLastConnect()`,
- `const char* meta_user_getRootMetaChunkHash()`,
- `const char* meta_user_getLocalChunkStoragePath()`.

Funkce `meta_user_getName` vrací identifikátor uživatele, `meta_user_getStorageSize` vrací velikost místa v úložišti, které má uživatel přidělené, `meta_user_getFreeStorageSize` vrací velikost volného místa v úložišti, `meta_user_getLastConnect` vrací datum a čas poslední provedené změny metadat, `meta_user_getRootMetaChunkHash` vrací hash uživatelského chunku metadat a `meta_user_getLocalChunkStoragePath` vrací místo, kam jsou ukládány a odkud jsou načítány chunky metadat.

Settery slouží k nastavení hodnot jednotlivých proměnných objektu `metaUser`. Tyto hodnoty jsou předávány v parametru dané funkce. Každý setter vytváří nový chunk s uživatelskými informacemi a informace o vytvořeném chunku jsou klientské aplikaci předávány ve struktuře `HashChunk`. Mezi settery patří funkce:

- `HashChunk meta_user_setName(const char* _userName)`,
- `HashChunk meta_user_setStorageSize(uint64_t _newSize)`,
- `HashChunk meta_user_setFreeStorageSize(uint64_t _newSize)`,
- `HashChunk meta_user_setLastConnect(const char* _lastConnect)`,
- `HashChunk meta_user_setRootMetaChunkHash(const char* _rootMetaChunkHash)`,
- `HashChunk meta_user_setLocalChunkStoragePath(const char* _localChunkStoragePath)`.

Funkce `meta_user_setName` nastaví identifikátor uživatele, `meta_user_setStorageSize` nastaví velikost místa v úložišti, které má uživatel přidělené, `meta_user_setFreeStorageSize` nastaví velikost volného místa v úložišti a `meta_user_setLastConnect` nastaví datum a čas poslední provedené změny dat. Funkce `meta_user_setRootMetaChunkHash` slouží k nastavení identifikátoru uživatelského chunku metadat. Tato funkce by však neměla být volána z klientské aplikace, protože veškeré nastavování identifikátorů chunků metadat je prováděno v rámci této knihovny. Funkce `meta_user_setLocalChunkStoragePath` slouží k nastavení místa, které slouží k výměně chunků metadat mezi touto knihovnou a klientskou aplikací.

6.6.4. Funkce pracující s metadaty

Tyto funkce slouží k základní práci s metadaty. Dělí se do dvou kategorií. V první kategorii jsou funkce, které pouze vracejí informace uložené v metadatech a druhou kategorii tvoří funkce, které buď do metadat ukládají nové informace, anebo naopak informace z metadat odstraňují. Při každé změně metadat dojde k vytvoření nových chunků metadat, ve kterých nastala změna. V tuto chvíli nastává trochu neoptimální situace v tom, že pokud je trochu složitější struktura chunků metadat, tedy pokud existují nějaké další chunky metadat a sdílené chunky metadat, tak dochází k vytváření více chunků metadat, než je třeba. Důvodem je to, že pokud máme například jeden uživatelský chunk metadat a jeden další chunk metadat a ke změně dojde v dalším chunku metadat, je vytvořen nový další chunk metadat, který má jinou hash, protože jeho obsah je jiný. Ale protože se změnila jeho hash, je třeba změnit tuto hash i v

uživatelském chunku metadat, což způsobí změnu jeho obsahu, a tudíž je vytvořen i nový uživatelský chunk metadat, který má jinou hash. Tuto hash je potom třeba nastavit v chunku s uživatelskými informacemi. To sice způsobí změnu obsahu, nicméně ne změnu hashe, protože u tohoto chunku se hash nemění. Takže místo jednoho chunku metadat jsou vytvořeny hned tři nové chunky. Toto řešení není optimální, nicméně není úplně špatné, protože kdyby měly chunky metadat pořád stejnou hash, dalo by se určit, že se jedná právě o chunky metadat. Nejvhodnějším řešením by byl mechanismus, který by zajistil kompromis mezi těmito řešeními, ale ten nebylo možné z časových důvodů navrhnout a realizovat.

Všechny vytvářené chunky metadat, mimo sdílených chunků metadat, jsou šifrovány uživatelským šifrovacím klíčem.

Pro práci s metadaty slouží následující funkce:

- `int meta_loadRootChunk(const char* _fileName)`,
- `int meta_loadChunk(const char* _fileName)`,
- `HashArray meta_createNewFile(const char* _name, uint64_t _size, int _replication, CChunkArray _chunks, const char* _path)`,
- `HashArray meta_createNewFolder(const char* _name, const char* _path)`,
- `HashArray meta_updateFile(uint64_t _size, CChunkArray _chunks, const char* _path)`,
- `HashArray meta_setFileReplication(int _replication, const char* _path)`,
- `HashChunk meta_getFileReplication(const char* _path)`,
- `HashChunk meta_getFileSize(const char* _path)`,
- `CChunkArray meta_getFileChunks(const char* _path)`,
- `FolderContentArray meta_getFolderContent(const char* _path)`,
- `HashArray meta_setFileName(const char* _path, const char* _newName)`,
- `HashArray meta_setFolderName(const char* _path, const char* _newName)`,
- `HashArray meta_deleteFile(const char* _path)`,
- `HashArray meta_deleteFolder(const char* _path)`.

Funkce **`meta_loadRootChunk`** slouží k načítání uživatelského chunku metadat, jehož hash je předána v parametru `_fileName`. Funkce nejprve tento chunk dešifruje použitím uživatelského šifrovacího klíče a uzly představující soubory či složky, které jsou obsaženy v tomto chunku, jsou do *stromu* přidávány jako následníci uzlu ***root***. Pokud vše proběhne v pořádku, funkce vrátí 0, jinak číslo menší než 0.

Funkce **`meta_loadChunk`** slouží k načítání dalších chunků metadat. Hash tohoto chunku je předána v parametru `_fileName`. Daný chunk je nejprve dešifrován pomocí uživatelského šifrovacího klíče. Následně je ve struktuře *stromu* vyhledán uzel, který obsahuje danou hash, a jako následníci tohoto uzlu jsou přidávány uzly představující soubory či složky, které jsou obsaženy v tomto chunku. Funkce vrací 0, pokud vše proběhne v pořádku, jinak číslo menší než 0.

`meta_createNewFile` je funkce, která v daném místě adresářové struktury, v případě knihovny v daném místě *stromu*, vytvoří nový soubor. Parametr `_name` předává jméno souboru, `_size` jeho celkovou velikost, `_replication` předává hodnotu určující počet replik daného souboru, resp. jeho chunků, které se v úložišti vytvoří. Parametr `_chunks` předává strukturu `CChunkArray`, která obsahuje informace o jednotlivých chuncích souboru. Parametr `_path` předává místo v adresářové struktuře, resp. ve *stromu*, kde bude nový soubor vytvořen. Tato funkce vrací strukturu `HashArray`, která obsahuje informace o vytvořených chuncích metadat.

`meta_createNewFolder` je funkce, která v daném místě adresářové struktury, v případě knihovny v daném místě *stromu*, vytvoří novou složku. V parametru `_name` je

předáváno jméno nové složky a parametr `_path` předává místo v adresářové struktuře, resp. ve *stromu*, kde bude nová složka vytvořena. Funkce vrací strukturu `HashArray`, která obsahuje informace o vytvořených chuncích metadat.

Funkce **`meta_updateFile`** slouží k aktualizaci daného souboru. Parametr `_size` předává celkovou velikost souboru, parametr `_chunks` předává informace o jednotlivých chuncích souboru a `_path` předává umístění v adresářové struktuře, resp. *stromu*, kde se daný soubor nachází. Funkce opět vrací informace o vytvořených chuncích metadat ve struktuře `HashArray`.

Funkce **`meta_setFileReplication`** nastaví pro daný soubor hodnotu určující počet replik daného souboru, resp. jeho chunků, které se v úložišti vytvoří. Umístění daného souboru je předáváno v parametru `_path` a hodnota určující počet replik je předávána v parametru `_replication`. Funkce vrací strukturu `HashArray` s informacemi o vytvořených chuncích metadat.

`meta_getFileReplication` má pouze parametr `_path`, který předává umístění daného souboru v adresářové struktuře, resp. *stromu*, jehož hodnotu počtu replikací má funkce vrátit. Tato hodnota je předávána ve struktuře `HashChunk`.

`meta_getFileSize` vrací celkovou velikost daného souboru ve struktuře `HashChunk`, jehož umístění v adresářové struktuře, resp. *stromu*, je předáváno v parametru `_path`.

Funkce **`meta_getFileChunks`** vrací ve struktuře `CChunkArray` informace o všech chuncích daného souboru, jehož umístění je předáno v parametru `_path`.

Funkce **`meta_getFolderContent`** vrací ve struktuře `FolderContentArray` obsah dané složky, tedy soubory a složky, které tato složka obsahuje. Umístění této složky je předáno v parametru `_path`.

`meta_setFileName` a **`meta_setFolderName`** jsou funkce sloužící k přejmenování souboru a složky. Umístění daného souboru či složky je předáváno parametrem `_path`. Parametr `_newName` předává nové jméno souboru nebo složky. Obě tyto funkce vracejí strukturu `HashArray`, která obsahuje informace o nově vytvořených chuncích metadat.

Funkce **`meta_deleteFile`** a **`meta_deleteFolder`** slouží k vymazání informací v metadatech o daném souboru či složce. Obě tyto funkce mají jeden parametr - `_path`, který určuje umístění daného souboru či složky v adresářové struktuře, resp. *stromu*, a vracejí strukturu `HashArray`, ve které jsou informace o vytvořených chuncích metadat.

6.6.5. Funkce zajišťující sdílení

Realizace sdílení probíhala na základě návrhu popsaného v předchozí kapitole. Bohužel z důvodu nedostatku času nemohl být realizován kompletní návrh, a proto nebyl realizován *změnový chunk* a rozlišování typu uživatelských práv pro daný sdílený soubor či sdílenou složku.

Mechanismus sdílení funguje tak, že uživatel vytvoří sdílený chunk metadat obsahující například sdílený soubor a do seznamu sdílejících uživatelů přidá nového sdílejícího uživatele spolu s typem přístupového práva k tomuto sdílenému souboru. To se však zatím nebude využívat. Funkce, která slouží k přidání nového uživatele, vrátí informaci o vytvořeném sdíleném chunku metadat - jeho hash a klíč, kterým je daný chunk zašifrován. Tyto dvě informace musí klientská část zaslat přidanému sdílejícímu uživateli, který si následně sdílený chunk metadat načte a uzel představující sdílený soubor, který je obsažen v tomto chunku, je do *stromu* přidán jako následník uzlu ***root***, tudíž sdílený soubor se sdílejícímu uživateli zobrazí přímo v kořeni adresářové struktury namapovaného síťového disku. Absence *změnového chunku* však způsobí, že pokud je provedena změna v tomto souboru, celý mechanismus se opakuje a uživateli, který daný soubor sdílel, se v kořeni adresářové struktury namapovaného síťového disku objeví ve-

dle původního sdíleného souboru i upravený sdílený soubor. Jednoduše řečeno úprava sdíleného souboru nepřepíše původní sdílený soubor, ale vytvoří nový.

Sdílení obstarávají následující funkce:

- `int meta_loadSharedChunk(const char* _fileName, const char* _key, int _fileOrFolder)`,
- `HashArray meta_setFileShared(const char* _path)`,
- `HashArray meta_setFolderShared(const char* _path)`,
- `HashArray meta_setFileNormal(const char* _path)`,
- `HashArray meta_setFolderNormal(const char* _path)`,
- `HashArray meta_addUserSharedFile(const char* _path, UserPerm _user)`,
- `HashArray meta_addUserSharedFolder(const char* _path, UserPerm _user)`,
- `UserPermArray meta_getUsersSharedFile(const char* _path)`,
- `UserPermArray meta_getUsersSharedFolder(const char* _path)`,
- `HashArray meta_updateUserSharedFile(const char* _path, UserPerm _user)`,
- `HashArray meta_updateUserSharedFolder(const char* _path, UserPerm _user)`,
- `HashArray meta_deleteUserSharedFile(const char* _path, UserPerm _user)`,
- `HashArray meta_deleteUserSharedFolder(const char* _path, UserPerm _user)`.

Funkce **`meta_loadSharedChunk`** slouží k načítání sdílených chunků metadat, jehož hash je předána v parametru `_fileName`. Daný sdílený chunk metadat je nejprve dešifrován pomocí šifrovacího klíče, který je předán v parametru `_key`, a uzly představující soubory či složky, které jsou obsaženy v tomto sdíleném chunku, jsou do *stromu* přidávány jako následníci uzlu ***root***. Ještě je třeba dodat, že pokud existuje následník uzlu ***root***, který má stejné jméno jako v přidávaném uzlu, je jméno v přidávaném uzlu rozšířeno o takové číslo, které nezpůsobí kolizi. Parametr `_fileOrFolder` pouze určuje, zda je ve sdíleném chunku metadat složka nebo soubor. Pokud vše proběhne v pořádku, funkce vrátí 0, jinak číslo menší než 0.

Funkce **`meta_setFileShared`** a **`meta_setFolderShared`** slouží k nastavení sdílení daného souboru či složky. Umístění daného souboru či složky v adresářové struktuře, resp. *stromu*, je předáváno v parametru `_path`. Tato funkce způsobí, že daný soubor či složka jsou přesunuty do samostatného sdíleného chunku metadat. Zároveň je náhodně vygenerován 128bitový šifrovací klíč, kterým je daný sdílený chunk šifrován. Obě tyto funkce vrací strukturu `HashArray`, která obsahuje informace o vytvořených chuncích metadat. Navíc je v této struktuře v informaci o sdíleném chunku předán i klíč, kterým je onen sdílený chunk zašifrován.

Funkce **`meta_setFileNormal`** a **`meta_setFolderNormal`** zruší sdílení daného souboru či složky. Obě tyto funkce mají jeden parametr - `_path`, který předává umístění daného sdíleného souboru či sdílené složky v adresářové struktuře, resp. *stromu*. Tyto funkce nastaví daný sdílený soubor nebo sdílenou složku jako normální a odstraní všechny sdílející uživatele. Tyto funkce vrací strukturu `HashArray`, která obsahuje informace o vytvořených chuncích metadat.

`meta_addUserSharedFile` a **`meta_addUserSharedFolder`** jsou funkce, které přidají nového sdílejícího uživatele, resp. jeho identifikátor, s daným typem přístupového práva do seznamu sdílejících uživatelů pro daný sdílený soubor či sdílenou složku. Parametr `_path` předává umístění daného sdíleného souboru nebo sdílené složky v adresářové struktuře, resp. *stromu*. Parametr `_user` předává strukturu `UserPerm`, která obsahuje identifikátor uživatele a typ přístupového práva pro daný sdílený soubor či sdílenou složku. Obě tyto funkce vrací strukturu `HashArray`, která obsahuje informace o vytvořených chuncích metadat a ve které je v informaci o sdíleném chunku předán i klíč, kterým je onen sdílený chunk šifrován.

Funkce **meta_getUsersSharedFile** a **meta_getUsersSharedFolder** vrací ve struktuře UserPermArray seznam sdílejících uživatelů, resp. jejich identifikátorů, spolu s typem přístupového práva pro daný sdílený soubor či sdílenou složku. Umístění sdíleného souboru či sdílené složky v adresářové struktuře, resp. *stromu*, je předáváno v parametru _path.

meta_updateUserSharedFile a **meta_updateUserSharedFolder** jsou funkce sloužící ke změně typu přístupového práva daného uživatele pro daný sdílený soubor či sdílenou složku. Parametr _path předává umístění daného sdíleného souboru nebo sdílené složky v adresářové struktuře, resp. *stromu*. Parametr _user předává strukturu UserPerm, která obsahuje identifikátor uživatele, jehož přístupové právo má být změněno, a nový typ přístupového práva. Obě tyto funkce vrací strukturu HashArray, která obsahuje informace o vytvořených chuncích metadat. V této struktuře je v informaci o sdíleném chunku předán i klíč, kterým je onen sdílený chunk šifrován.

Funkce **meta_deleteUserSharedFile** a **meta_deleteUserSharedFolder** slouží k odstranění sdílejícího uživatele ze seznamu sdílejících uživatelů pro daný sdílený soubor či sdílenou složku. Umístění sdíleného souboru nebo sdílené složky v rámci adresářové struktury, resp. *stromu*, je předáváno v parametru _path. Identifikátor uživatele, který má být odstraněn, je obsažen ve struktuře UserPerm, která je předávána parametrem _user. Tyto funkce vrací strukturu HashArray, která obsahuje informace o vytvořených chuncích metadat. Obě tyto funkce vrací strukturu HashArray, která obsahuje informace o vytvořených chuncích metadat a ve které je v informaci o sdíleném chunku předán i klíč, kterým je onen sdílený chunk šifrován.

6.7. Historie

V předchozí kapitole byl vytvořen návrh historie, která by obsahovala informace o provedených změnách v adresářové struktuře a zajišťovala verzování souborů. Nicméně s její realizací se od začátku nepočítalo, protože by její realizace byla dost časově náročná a protože by stejně nebylo k dispozici rozhraní, skrze které by se s historií pracovalo. Avšak bylo s jejím případným budoucím nasazením počítáno.

6.8. Udržování velikosti chunků metadat

Jak již bylo popsáno v předchozí kapitole, není vhodné, aby chunky metadat mohly mít jakoukoliv velikost. Proto byl na základě návrhu popsaného v předchozí kapitole vytvořen mechanismus, který se stará o udržování velikosti chunků metadat.

Protože známe jednotlivé typy uzlů (typy souborů a složek) ve struktuře *stromu* a pro každý typ uzlu známe jeho strukturu ve formátu JSON, je možné pro každý uzel spočítat velikost v B, kterou bude zabírat ve výsledném chunku metadat. Tento mechanismus realizován na základě rekurzivního algoritmu, který před každým vytvářením chunků metadat spočte velikosti jednotlivých objektů (souborů a složek) ve formátu JSON. Pokud během výpočtu překročí objekt představující složku maximální velikost, je v této složce vybrána položka mající největší velikost, která je následně přesunutá do samostatného chunku metadat. Pokud objekt představující soubor překročí během výpočtu maximální velikost, je tento soubor přesunut do samostatného chunku metadat. Celý tento algoritmus je realizován ve třídě MetaStructure v metodě **prepareNodesForOutput**. V tuto chvíli je maximální velikost chunku metadat nastavena na 64 kB (65536 B).

Tento algoritmus je dobrý, ale není dokončený, protože k jeho dokončení chyběl potřebný čas. Struktura metadat ve formátu JSON byla navržena tak, že i objekt, který

představuje odkaz na další chunk, obsahuje několik informací a to znamená, že v chunku metadat zabírá nějakou velikost. Pokud nastane situace, že například objekt představující složku překročí maximální velikost a uvnitř tohoto objektu jsou pouze položky představující odkazy na další chunky metadat, je toto překročení ignorováno, protože neexistuje položka, kterou by bylo možné přesunout do samostatného chunku metadat a zároveň zachovat adresářovou strukturu. Aby k této situaci nedocházelo, je třeba strukturu chunků metadat rozšířit o možnost zřetězení chunků metadat, která by tuto situaci vyřešila.

6.9. Implementace zabezpečení systému

V této části je popsána implementace zabezpečení celého systému, která vychází z návrhu popsaného v předchozí kapitole.

6.9.1. Šifrování dat

Funkce sloužící k šifrování dat jsou implementovány v knihovně `libsfuncs.a`. Konkrétně se jedná o funkci `encryptFileAES128CTR`, která pro šifrování dat používá algoritmus AES-128 v módu CTR, a funkci `decryptFileAES128CTR`, která slouží k dešifrování dat zašifrovaných funkcí `encryptFileAES128CTR`. Obě tyto funkce jsou podrobněji popsány v části 6.5.

Uživatelský 128bitový šifrovací klíč je vytvořen tak, že z části uživatelova primárního klíče je vytvořena hash, ze které je jako šifrovací klíč použito prvních 32 znaků (128 bitů). Podrobnější informace o tvorbě uživatelského 128bitového šifrovacího klíče jsou v diplomové práci Jana Janury [2].

Ke generování 128bitových šifrovacích klíčů sloužících k šifrování sdílených chunků metadat slouží funkce `generateAES128Key`, která je implementována v knihovně `libsfuncs.a`. Podrobněji je tato funkce popsána v části 6.5.

6.9.2. Zabezpečení komunikace

Pro zabezpečení komunikace pomocí protokolu TLS, který používá *OpenPGP* certifikáty, byla použita knihovna *GnuTLS* [57]. Tato knihovna funguje na většině Unixových platformách i na Windows. Aby bylo možné využívat funkce poskytované touto knihovnou, je třeba nainstalovat tyto balíčky:

- *m4*,
- *gmp-5.1.3*,
- *nettle-2.7.1*,
- *gnutls-3.1.0*,
- *libgnutls-dev* a
- *libgnutls28*.

Pro implementaci zabezpečení komunikace protokolem TLS s použitím knihovny *GnuTLS* byly použity části zdrojových kódů popsané v článku *Implementace v GnuTLS* [58]. Při implementaci TLS pomocí *GnuTLS* nebyly zatím pro jednotlivé fáze TLS použity konkrétní šifrovací algoritmy, ale bylo použito makro definující množinu šifer. To funguje tak, že obě komunikující strany se v první fázi TLS domluví na konkrétních šifrovacích algoritmech z této množiny používaných v jednotlivých fázích TLS. Toto makro je v tzv. *priority strings* definováno klíčovým slovem NORMAL. Podrobnější informace o tomto makru jsou k nalezení v dokumentaci *GnuTLS* [59].

Pár *OpenPGP* klíčů, který protokol TLS využívá, byl vygenerován pomocí aplikace *Hesla a klíče*, která je integrována přímo v Ubuntu. Podrobný návod, jak v této aplikaci vygenerovat *OpenPGP* klíčový pár, je v článku *How do I make a PGP key?* [60]. Ještě je třeba dodat, že vygenerované klíče byly typu RSA délky 1024 bitů.

7. Testování

Tato kapitola popisuje nejprve testování samotného subsystému pro správu metadat a následně se věnuje testování pilotní implementace celého systému.

7.1. Testování subsystému pro správu metadat

Víceméně se dá říci, že testování subsystému pro správu metadat bylo rozděleno do třech fází. V první fázi testování byla nejprve otestována funkčnost jednotlivých funkcí knihovny a následně byly simulovány jednotlivé situace. Simulace těchto situací probíhala postupným voláním jednotlivých funkcí knihovny, které by za dané situace byly volány, a následnou kontrolou uložených informací a návratových hodnot.

Ve druhé fázi testování byl subsystém pro správu metadat otestován ve spojení s klientskou aplikací. V této fázi testování bylo odhaleno několik chyb a implementačních nedostatků, které nebyly odhaleny v první fázi. Navíc se implementace některých funkcí ukázaly jako nevhodné pro použití v klientské aplikaci a musely být upraveny.

V poslední fázi testování byl subsystém pro správu metadat otestován v rámci celého systému. V této fázi však už nebyla objevena žádná další chyba.

7.2. Testování v rámci celého systému

Celý systém byl otestován dvakrát, protože při prvním testu byla odhalena závažná chyba v klientské aplikaci. Tato chyba byla záhy odstraněna a systém byl otestován znovu.

7.2.1. Testovací sestava

Testování celého systému probíhalo na sestavě, která je zobrazena na Obr. 14. Tuto sestavu tvořily čtyři virtuální servery, na kterých běžel subsystém pro správu dat, jeden virtuální server, který sloužil jako přístupový server, a osobní počítač, na kterém byla spuštěna klientská aplikace využívající subsystém pro správu metadat.

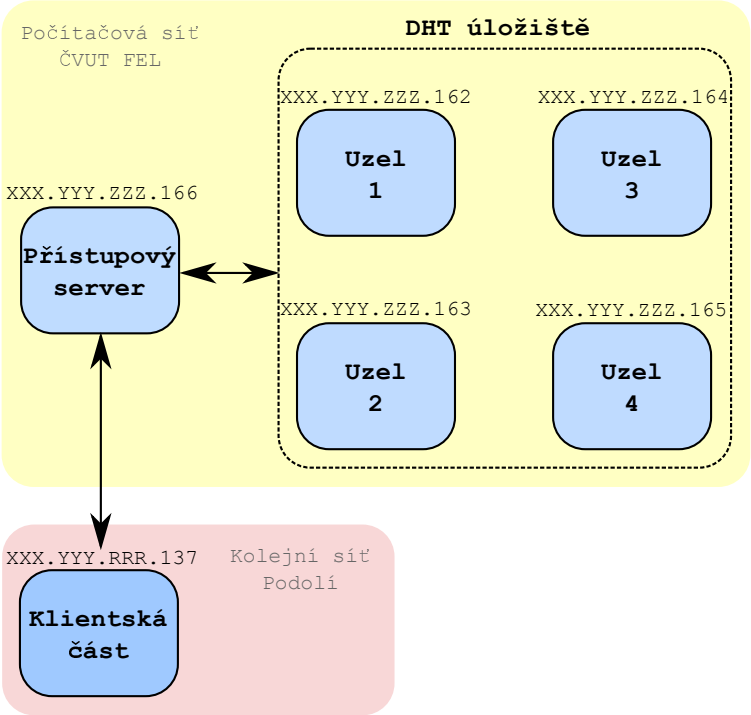
Virtuální servery byly umístěny na serveru v síti ČVUT FEL na Karlově náměstí. Parametry hostujícího serveru byly následující:

- Hardware
  - Typ: Fujitsu Primergy RX300
  - Procesor: 2x Intel® Xeon CPU E5-2620 v2 @ 2.10GHz
  - Síťové připojení: 2x 10 GbE
  - HDD: 4x 600 GB 3.5" @ 15k SAS (RAID 5)SW
- Software
  - Linux cn01 3.2.0-4-amd64 #1 SMP Debian 3.2.51-1 x86_64 GNU/Linux
  - QEMU emulator version 1.1.2 (qemu-kvm-1.1.2+dfsg-6, Debian), Copyright © 2003-2008 Fabrice Bellard

– libvirtd (libvirt) 0.9.12.3

Parametry virtuálních serverů byly následující:
- 1x VCPU
- RAM: 0,5 GB
- HDD: 20 GB
- OS: Debian 7.0.3 64bit

Osobní počítač byl umístěn v kolejní síti Podolí se symetrickým připojením 1 Gb/s. Parametry osobního počítače byly následující:
- Procesor: Intel® Core™ 2 Duo T6500 @ 2,1 GHz
- RAM: 4 GB DDR2
- HDD: 20 GB
- OS: Ubuntu 12.04 LTS 64bit



Obrázek 14. Sestava pro otestování celého systému.

7.2.2. Výsledky měření

V prvním testu byl celý systém otestován s malými soubory. Všechny soubory se podařilo do úložiště úspěšně nahrát a následně úspěšně stáhnout. Tento test také ověřil správnou tvorbu replikací. Naměřené časy nahrávání jednotlivých souborů jsou zobrazeny v Tab. 13.

Při stahování nebyly soubory stahovány jednotlivě, ale byly staženy všechny nahrané soubory najednou. Celková velikost těchto souborů byla 18,6 MB a stahování trvalo 35s (průměrná rychlost downloadu byla 4,25 Mb/s). Struktura těchto souborů byla následující:

```
/
├── file.txt
├── directory1
│   ├── fotka1.jpg
│   └── fotka2.jpg
└── directory2
    ├── song.mp3
    └── file.zip
```

Soubor	Velikost	Doba uploadu	Průměrná rychlost uploadu [Mb/s]
Textový soubor `file.txt`	250 kB	12s	0,16
2 fotografie `directory1`	1,2 MB	13s	0,74
Zvukový soubor `song.mp3`	6,3 MB	15s	3,36
ZIP soubor `file.zip`	10,9 MB	15s	5,81

Tabulka 13. Výsledky měření pro první test celého systému.

V tomto testu byl celý systém otestován pouze s malými soubory, protože při nahrávání větších souborů nebo většího množství menších souborů najednou docházelo k nekorektnímu ukončování klientské aplikace. Tento problém byl způsobený špatnou detekcí ukončení zápisu do souboru a projevil se až při závěrečném testování. Na základě prvního testu bylo zjištěno, že TLS handshake značně zpomaluje celou komunikaci. Zpomalení komunikace navíc způsobovala i klientská aplikace, která operace prováděla sekvenčně.

Chybu v klientské aplikaci se podařilo úspěšně odstranit, a proto mohl být v druhém testu celý systém otestován s velkými soubory a s větším množství menších souborů. Naměřené časy nahrávání jednotlivých souborů jsou zobrazeny v Tab. 14.

Soubor	Velikost	Doba uploadu	Průměrná rychlost uploadu [Mb/s]
12 fotografií `directory1`	23,4 MB	5m	0,62
ZIP soubor `file1.zip`	23,1 MB	1m	3,00
ZIP soubor `file2.zip`	51,3 MB	1m 30s	4,56
ZIP soubor `file3.zip`	335,4 MB	4m	11,18

Tabulka 14. Výsledky měření pro druhý test celého systému.

Při stahování byly opět stahovány všechny soubory najednou. Celková velikost stahovaných souborů byla 434,3 MB a doba stahování byla 7m 30s (průměrná rychlost

downloadu byla 7,72 Mb/s). Struktura těchto souborů byla následující:

```
/
├── directory1
│   ├── fotka1.jpg
│   ├── fotka2.jpg
│   ├── ...
│   └── fotka12.jpg
├── file1.zip
├── file2.zip
└── file3.zip
```

Měření v druhém testu ukázalo, že nahrávání většího množství souborů trvá déle než nahrávání 15krát většího souboru. Tento fakt je způsoben již zmíněným způsobem provádění operací v klientské aplikaci.

Ačkoliv se první test může jevit spíše jako neúspěšný než úspěšný, opak je pravdou. Díky chybě v klientské aplikaci byla otestována spolehlivost datového úložiště, protože klientská aplikace zasílala datovému úložišti zprávy buď ve špatném formátu, nebo se špatnými parametry, anebo došlo k neočekávanému ukončení spojení. Datové centrum však zůstalo dále provozuschopné. Oba testy odhalily implementační nedostatky systému, které je třeba odstranit. V první řadě je třeba zefektivnit práci klientské aplikace, zefektivnit tvorbu souborů s metadaty a zoptimalizovat komunikaci protokolem TLS.

8. Závěr

Cílem této práce bylo v distribuovaném úložišti dat navrhnout a následně implementovat subsystém pro správu metadat. Před samotným návrhem subsystému byla provedena analýza několika současných cloudových úložišť, které byly následně podrobeny několika testům. Na základě mé analýzy a analýz Martina Kudrnáče [1] a Jana Janury [2] byla vytvořena tabulka, která porovnává nejznámější současná cloudová úložiště. Z této analýzy vyplynuly požadavky na subsystém pro správu metadat. Na základě těchto požadavků byl sestaven návrh subsystému a dle tohoto návrhu byla následně provedena pilotní implementace subsystému. Vytvořený subsystém pro správu metadat byl poté propojen s klientskou aplikací, která je výstupem práce Jana Janury [2], a subsystémem pro správu dat, který je výstupem práce Martina Kudrnáče [1]. Propojením všech těchto částí vznikla pilotní implementace distribuovaného úložiště dat, která byla úspěšně otestována.

Pilotní implementace subsystému pro správu metadat splňuje zadání a subsystém poskytuje všechny potřebné funkce pro práci s metadaty. Z časových důvodů však nebylo možné realizovat některé funkce zmíněné v návrhu a některé funkce byly oproti původnímu návrhu trochu zjednodušeny. Avšak se všemi funkcemi z návrhu bylo při implementaci počítáno a tudíž by se neměl vyskytnout závažnější problém při jejich dodatečné realizaci. Zároveň se podařilo úspěšně celý systém zabezpečit podle návrhu v kapitole 5.

Subsystému pro správu metadat poskytuje mnoho možností pro další rozvoj. Primárně je třeba dodělat funkce, které byly oproti původnímu návrhu trochu zjednodušeny. To znamená doplnit vytváření *změnového chunku* do funkce, která vytváří sdílené chunky metadat. Dále pak je možné subsystém rozšířit o navrženou historii, optimalizovat tvorbu chunků metadat a zefektivnit komunikaci protokolem TLS.

Příloha A.

Porovnání cloudových úložišť

	Google Disk	SkyDrive	DropBox	Ubuntu One	SugarSync
Obecné					
Jazyková lokalizace	●	●	○	●	○
Integrace do kontextového menu	●	●	○	●	●
Vlastní nasazení	○	○	○	○	○
Webová aplikace	●	●	●	●	●
Klientská aplikace Windows / Linux / OS X / Ostatní	●/○/●/○	●/○ /●/○	●/●/●/○	●/●/●/○	●/○/●/○
Mobilní aplikace Android / Windows Phone / iOS / Ostatní	●/○/●/○	●/●/●/○	●/●/●/●	●/○/●/○	●/●/●/●[2]
Cloudové úložiště	●	●	●	●	●
Bezplatná kapacita [GB]	15	7	2	5	5
Maximální kapacita [GB]	16384	207	100	○	1000
Maximální velikost souboru [MB]	10240	2048	300[4]	○	-
Možnost rozšíření zdarma	○	○	○	○	●
Synchronizace					
Synchronizace libovolné složky	○	○	○	●	●
Synchronizace podsložek	●	●	●	○	●
Synchronizace jednotlivých souborů	○	○	○	○	●
Pozastavení synchronizace	●	○	●	●	○
Sledování přenosu dat	●[5]	○	○	○	●
Sdílení					
Veřejné	●	●	○	●	●
Odkazem	●	●	●	●	●
Pomocí pozvánky	●	●	●	●	●
Pomocí mailu	●	●	●	●	●
Skupinové	●	●	○	○	●
Nastavení oprávnění	●	●	○	●	○
Zabezpečení					
Šifrování na straně serveru	○	○	AES	○	AES-128
Šifrovaní na straně klienta	○	○	○	○	○
Šifrování přenosu	SSL	SSL	SSL	SSL	SSL (3.3) + TLS
Přihlášení pomocí dvoufázového ověření	●	○	●	○	○
Další funkce					
Obnova smazaných souborů	●	●	●	60 dnů[11]	○
Verzování	●	●[6]	●	○	●
Uložení verzí	30 dnů[7]	25 verzí	30 dnů	○	30 dnů
Nastavení rychlosti uploadu	○	○	●	●	●[9]
Nastavení priorit pro upload	○	○	○	○	●
Nastavení rychlosti downloadu	○	○	●	●	○
Rychlosti					
Rychlost uploadu malých souborů [Mb/s]	0,05	0,23	0,25	0,058	0,04
Doba uploadu malých souborů	3h 57m	4h 12m	15m	2h 16m	1h 50m
Rychlost uploadu středně velkých souborů [Mb/s]	4,77	2,83	5,56	9,65	7,15
Doba uploadu středně velkých souborů	14m	21m 31s	5m	2m 30s	7m 15s
Rychlost uploadu velkého souboru [Mb/s]	21,2	4	0,127	0,32	-
Doba uploadu velkého souboru	10m	58m	90s	3m	-
Rychlost downloadu malých souborů [Mb/s]	0,08	0,621	-	-	0,33
Doba downloadu malých souborů	2h 18m	1h 15m	-	-	5h 13m
Rychlost downloadu středně velkých souborů [Mb/s]	26,19	5,315	26,31	-	2,67
Doba downloadu středně velkých souborů	3m	51m	2m 30s	-	30m 41s
Rychlost downloadu velkého souboru [Mb/s]	58,92	2,8	53,77	11,8	2,9
Doba downloadu velkého souboru	4m	1h 15m	3m	18m	1h 20m
	Google Disk	**SkyDrive**	**DropBox**	**Ubuntu One**	**SugarSync**

● Ano
○ Ne
- Nezjištěno/Nenaměřeno

1 - FreeBSD
2 - BlackBerry
3 - Společný úložný prostor pro služby Google Disk, Gmail a Fotky Google+
4 - Pouze přes webové rozhraní

Tato tabulka vznikla ve spolupráci s Martinem Kudrnáčem [1] a Janem Janurou [2]. Zobrazuje parametry a výsledky testů všech testovaných cloudových úložišť.

	MEGA	TeamDrive	Wuala	cloudMe	SpiderOak	BitTorrentSync	ownCloud
	●	○	●	○	○	○	●
	○	○	○	●	●	○	○
	○	●	○	○	○	●	●
	●	○	○	●	●	○	●
	●/(○/○/○)[10]	●/●/●/○	●/●/●/○	●/●/●/○	●/●/●/○	●/●/●/●[1]	●/●/●/○
	●/○[11]/●/●[2]	●/○/●/○	●/○/●/○	●/○/●/○	●/○/●/○	●/○/●/○	●/○/●/○
	●	●	●	●	●	○	○
	50	2	5	3	2	○	○
	4096	neomezeno	2048	500	100	○	○
	neomezeno	-	40960	150	-	○	○[12]
	○	●	●	○	●	○	○
	●	●	●	●	●	●	●
	●	○	●	○	●	○	○
	●	○	●	○	●	○	○
	●	●	●	●	●	●	○
	●	●	●	●	●	●	●
	●	○	●	●	●	○	○
	●	●	●	●	●	○	●
	○	●	○	●	○	○	○
	○	●	○	●	○	○	●
	●	●	●	●	●	●	○
	●	●	●	○	●	●	○
	○	○	AES-128	○	AES-256	○	●
	RSA-2048, AES-128	RSA-2048, AES-256	RSA-2048, AES-128	○	RSA-3076, AES-256	AES-128	○
	SSL	○	SSL	SSL	SSL	○	○
	○	○	○	○	○	○	○
	●	●	○	●	○	30 dnů[13]	●
	○	●	●	○	●	●	●
	○	neomezeno	10 verzí	-	-	30 dnů[8]	-
	●	○	●	●	●	●	●
	○	○	●	○	○	○	●
	○	○	●	●	○	●	●
	-	1,17	0.7	0,1	0,5	2,34	0,012
	-	10m	30m	3h 55m	12m	5m	1d 7h 20m
	1,32	8,6	2,2	4,46	1,358	22,9	4,93
	51m	8m	13m 13s	17m	1h 4m	3m	13m
	63,28	9,09	28,5	-	1,483	33,5	28,92
	3m	22m	6m	-	2h 40m	6m	7m
	-	1,31	2	-	0,4	2,3	2,68
	-	9m	7m	-	15m	5m	1m 30s
	6,34	20,35	10,8	19,89	2,28	22,64	26,94
	11m	4m	6m 58s	2m 30s	33m 44s	3m	3m
	35,6	10,98	7,3	-	2,45	28,61	43
	6m	18m	30m 41s	-	2h 18m	7m	5m
	MEGA	TeamDrive	Wuala	cloudMe	SpiderOak	BitTorrentSync	ownCloud

5 - Pouze počet přenesených složek a souborů
6 - Pouze Office soubory
7 - Verze souborů z aplikací Google Dokumenty, Tabulky a Prezentace jsou ukládány navždy
8 - Ve výchozím nastavení, lze změnit
9 - Pouze relativní nastavení
10 - Zatím ve vývoji (17.11.2013)
11 - Bez 100% garance
12 - Přes webové rozhraní 8MB, jinak podle konfigurace
13 - Dobu lze nastavit

Příloha B.

Instalační a uživatelská příručka

B.1. Prostředí pro běh

Knihovna pro správu metadat - `libMetaManagement.so` byla vyvinuta a testována na operačních systémech Ubuntu 12.04 LTS 32bit a 64bit. Tuto knihovnu však lze použít i na platformě Windows. Knihovna `libMetaManagement.so` využívá některé funkce knihovny *OpenSSL*, která je dostupná jak pro Linux, tak pro Windows.

B.2. Kompilace

Nejprve je třeba na příslušné platformě vytvořit *statickou knihovnu* `libsfuncs.a`, to znamená vytvořit *statickou knihovnu* ze zdrojových souborů projektu **sfuncs**. Poté je třeba na příslušné platformě vytvořit dynamickou knihovnu `libMetaManagement.so`, to znamená s použitím `g++` kompilátoru zkompilovat jako *dynamickou knihovnu* zdrojové soubory projektu **MetaManagement** spolu s nalinkovanou knihovnou `libsfuncs.a`.

B.3. Použití

Použití knihoven `libMetaManagement.so` a `libsfuncs.a` je podobné. Obě tyto knihovny je třeba nalinkovat do příslušného projektu a poté už je možné používat všechny jejich funkce. Celý projekt je pak třeba zkompilovat spolu s nalinkovanou knihovnou.

B.4. Seznam metod a funkcí

B.4.1. Knihovna `libsfuncs.a`

- `void genHashSha256_64ext(char* _output, char* _filePath, char* _username, char* _timestamp);`
- `void genPrivateKeyHash(char* _output, char* _filePath);`
- `void getCurrentTime(char* _output);`
- `int encryptFileAES128CTR(const char* _readFilePath, const char* _writeFilePath, const char* _hexKey);`
- `int decryptFileAES128CTR(const char* _readFilePath, const char* _writeFilePath, const char* _hexKey);`
- `void generateAES128Key(unsigned char* _key);`
- `void convertHexToByte(unsigned char* _bytes, const char* _hex);`
- `void convertByteToHex(char* _hex, unsigned char* _bytes);`

B.4.2. Knihovna `libMetaManagement.so`

- void **meta_Init**(const char* _key, const char* _localChunkStoragePath, const char* _userChunkHash);
- void **meta_Exit**();
- HashArray **meta_firstLaunch**(const char* _username, uint64_t _storageSize, const char* _lastConnect);
- int **meta_user_loadChunk**(const char* _fileName);
- HashChunk **meta_user_setName**(const char* _userName);
- const char* **meta_user_getName**();
- HashChunk **meta_user_setStorageSize**(uint64_t _newSize);
- uint64_t **meta_user_getStorageSize**();
- HashChunk **meta_user_setFreeStorageSize**(uint64_t _newSize);
- uint64_t **meta_user_getFreeStorageSize**();
- HashChunk **meta_user_setLastConnect**(const char* _lastConnect);
- const char* **meta_user_getLastConnect**();
- HashChunk **meta_user_setRootMetaChunkHash**(const char* _rootMetaChunkHash);
- const char* **meta_user_getRootMetaChunkHash**();
- HashChunk **meta_user_setLocalChunkStoragePath**(const char* _localChunkStoragePath);
- const char* **meta_user_getLocalChunkStoragePath**();
- int **meta_loadRootChunk**(const char* _fileName);
- int **meta_loadChunk**(const char* _fileName);
- int **meta_loadSharedChunk**(const char* _fileName, const char* _key, int _fileOrFolder);
- HashArray **meta_createNewFile**(const char* _name, uint64_t _size, int _replication, CChunkArray _chunks, const char* _path);
- HashArray **meta_createNewFolder**(const char* _name, const char* _path);
- HashArray **meta_updateFile**(uint64_t _size, CChunkArray _chunks, const char* _path);
- HashArray **meta_setFileReplication**(int _replication, const char* _path);
- HashChunk **meta_getFileReplication**(const char* _path);
- HashChunk **meta_getFileSize**(const char* _path);
- CChunkArray **meta_getFileChunks**(const char* _path);
- FolderContentArray **meta_getFolderContent**(const char* _path);
- HashArray **meta_setFileName**(const char* _path, const char* _newName);
- HashArray **meta_setFolderName**(const char* _path, const char* _newName);
- HashArray **meta_setFileShared**(const char* _path);
- HashArray **meta_setFolderShared**(const char* _path);
- HashArray **meta_setFileNormal**(const char* _path);
- HashArray **meta_setFolderNormal**(const char* _path);
- HashArray **meta_addUserSharedFile**(const char* _path, UserPerm _user);
- HashArray **meta_addUserSharedFolder**(const char* _path, UserPerm _user);
- UserPermArray **meta_getUsersSharedFile**(const char* _path);
- UserPermArray **meta_getUsersSharedFolder**(const char* _path);
- HashArray **meta_updateUserSharedFile**(const char* _path, UserPerm _user);
- HashArray **meta_updateUserSharedFolder**(const char* _path, UserPerm _user);
- HashArray **meta_deleteUserSharedFile**(const char* _path, UserPerm _user);
- HashArray **meta_deleteUserSharedFolder**(const char* _path, UserPerm _user);
- HashArray **meta_deleteFolder**(const char* _path);

- `HashArray meta_deleteFile(const char* _path);`

Příloha C.

Obsah přiloženého CD

```
/
├── README
└── text
    ├── Dyntar_Tomas-DP-2014.pdf
    └── source files
        ├── abbreviations.tex
        ├── abstract.tex
        ├── acknowledgement.tex
        ├── app01.tex
        ├── app02.tex
        ├── app03.tex
        ├── declaration.tex
        ├── dyntatom.bib
        ├── dyntatom.pdf
        ├── dyntatom.tex
        ├── felthesis.cls
        ├── ch01.tex
        ├── ch02.tex
        ├── ch03.tex
        ├── ch04.tex
        ├── ch05.tex
        ├── ch06.tex
        ├── ch07.tex
        ├── ch08.tex
        ├── lev.pdf
        ├── meta1-eps-converted-to.pdf
        ├── meta2-eps-converted-to.pdf
        ├── meta3-eps-converted-to.pdf
        ├── sdilenaslozka1-eps-converted-to.pdf
        ├── sdilenaslozka2-eps-converted-to.pdf
        ├── sdileni1-eps-converted-to.pdf
        ├── sdileni2-eps-converted-to.pdf
        ├── sdilenysoubor1-eps-converted-to.pdf
        ├── sdilenysoubor2-eps-converted-to.pdf
        ├── sluzby1.jpg
        ├── sluzby2.jpg
        ├── spolecnynavrh-eps-converted-to.pdf
        ├── testovani-eps-converted-to.pdf
        ├── uzivatel-eps-converted-to.pdf
        └── vicechunku-eps-converted-to.pdf
```

```
│
│   └── vysvetlivky-eps-converted-to.pdf
└── project
    ├── libraries
    │   ├── libMetaManagement.so
    │   └── libsfuncs.a
    └── source files
        └── MetaManagement
            └── Debug
                ├── src
                │   ├── Jzon
                │   │   ├── Jzon.d
                │   │   └── subdir.mk
                │   ├── MetaManipulation
                │   │   ├── FileHash
                │   │   │   ├── FileHash.d
                │   │   │   ├── FileHash.o
                │   │   │   └── subdir.mk
                │   │   ├── Jzon v2-1
                │   │   │   ├── Jzon.d
                │   │   │   ├── Jzon.o
                │   │   │   └── subdir.mk
                │   │   ├── MetaStructure
                │   │   │   ├── Chunk.d
                │   │   │   ├── Chunk.o
                │   │   │   ├── MetaStructure.d
                │   │   │   ├── MetaStructure.o
                │   │   │   ├── Node.d
                │   │   │   ├── Node.o
                │   │   │   ├── UserRight.d
                │   │   │   ├── UserRight.o
                │   │   │   └── subdir.mk
                │   │   ├── MetaUser
                │   │   │   ├── MetaUser.d
                │   │   │   ├── MetaUser.o
                │   │   │   └── subdir.mk
                │   │   ├── MetaManipulation.d
                │   │   ├── MetaManipulation.o
                │   │   └── subdir.mk
                │   ├── CObjects.d
                │   ├── CObjects.o
                │   ├── Main.d
                │   ├── Main.o
                │   ├── MetaManagement.d
                │   ├── MetaManagement.o
                │   └── subdir.mk
                ├── libMetaManagement.so
                ├── makefile
                ├── MetaManagement
                ├── MetaManagement-v0
                └── objects.mk
```

```
│
│   └── sources.mk
│   ├── lib
│   │   └── libsfuncs.a
│   ├── src
│   │   ├── MetaManipulation
│   │   │   ├── FileHash
│   │   │   │   ├── FileHash.cpp
│   │   │   │   └── FileHash.h
│   │   │   ├── Jzon v2-1
│   │   │   │   ├── Jzon.cpp
│   │   │   │   ├── Jzon.h
│   │   │   │   ├── license.txt
│   │   │   │   └── README
│   │   │   ├── MetaStructure
│   │   │   │   ├── Chunk.cpp
│   │   │   │   ├── Chunk.h
│   │   │   │   ├── MetaStructure.cpp
│   │   │   │   ├── MetaStructure.h
│   │   │   │   ├── Node.cpp
│   │   │   │   ├── Node.h
│   │   │   │   ├── UserRight.cpp
│   │   │   │   └── UserRight.h
│   │   │   ├── MetaUser
│   │   │   │   ├── MetaUser.cpp
│   │   │   │   └── MetaUser.h
│   │   │   ├── MetaManipulation.cpp
│   │   │   └── MetaManipulation.h
│   │   ├── CObjects.cpp
│   │   ├── CObjects.h
│   │   ├── MetaManagement.cpp
│   │   └── MetaManagement.h
└── sfuncs
    ├── Debug
    │   ├── src
    │   │   ├── libsfuncs.d
    │   │   ├── libsfuncs.o
    │   │   ├── sfuncs.d
    │   │   ├── sfuncs.o
    │   │   └── subdir.mk
    │   ├── HashGen
    │   ├── libsfuncs
    │   ├── libsfuncs.a
    │   ├── makefile
    │   ├── objects.mk
    │   ├── sfuncs
    │   └── sources.mk
    └── Release
        ├── src
        │   ├── libsfuncs.d
        │   └── libsfuncs.o
```

```
│   │   │
│   │   ├── main.d
│   │   ├── main.o
│   │   ├── sfuncs.d
│   │   ├── sfuncs.o
│   │   └── subdir.mk
│   ├── makefile
│   ├── objects.mk
│   └── sources.mk
└── src
    ├── sfuncs.c
    └── sfuncs.h
```

Literatura

[1] Martin Kudrnáč. *Distribuované úložiště dat - správa dat.* Diplomová práce. Praha, květen 2014.

[2] Jan Janura. *Distribuované úložiště dat - klientská část.* Diplomová práce. Praha, květen 2014.

[3] *Distributed systems.* URL: http://www.answers.com/topic/distributed-system (cit. 30. 04. 2014).

[4] *CLOUD COMPUTING: CO TY POJMY ZNAMENAJÍ?* URL: http://cloud.cz/cloud/158-cloud-computingco-ty-pojmy-znamenaji.html (cit. 30. 04. 2014).

[5] *metadata.* URL: http://whatis.techtarget.com/definition/metadata (cit. 30. 04. 2014).

[6] *Google Drive.* URL: http://en.wikipedia.org/wiki/Google_Drive (cit. 02. 02. 2014).

[7] *Google Drive.* Google, Inc. URL: http://www.google.com/drive/about.html (cit. 02. 02. 2014).

[8] *Overview of Google Drive.* Google, Inc. URL: https://support.google.com/drive/answer/2424384?hl=en&ref_topic=14942&rd=1 (cit. 02. 02. 2014).

[9] *Storage plan pricing.* Google, Inc. URL: https://support.google.com/drive/answer/2375123?hl=en&ref_topic=2375005 (cit. 02. 02. 2014).

[10] *Google Docs, Sheets, and Slides size limits.* Google, Inc. URL: https://support.google.com/drive/answer/37603?hl=en&ref_topic=2375187 (cit. 02. 02. 2014).

[11] *Features.* Google, Inc. URL: http://www.google.com/drive/features.html (cit. 02. 02. 2014).

[12] *Zásady ochrany soukromí.* Google, Inc. URL: http://www.google.com/policies/privacy/ (cit. 02. 02. 2014).

[13] *Smluvní podmínky společnosti Google.* Google, Inc. URL: http://www.google.com/policies/terms/ (cit. 02. 02. 2014).

[14] *Syncdocs – Improved Google Drive Sync and Backup.* URL: http://www.syncdocs.com/ (cit. 02. 02. 2014).

[15] *Boxcryptor – Secure your Cloud.* URL: https://www.boxcryptor.com/en/boxcryptor-encryption-cloud-storage-window-mac-android-ios (cit. 02. 02. 2014).

[16] *Lepší zabezpečení účtu Google.* Google, Inc. URL: http://www.google.com/intl/cs/landing/2step/index.html (cit. 02. 02. 2014).

[17] *TeamDrive.* URL: http://en.wikipedia.org/wiki/TeamDrive (cit. 03. 02. 2014).

[18] *TeamDrive.* URL: http://www.bestbackups.com/blog/3747/teamdrive-review/ (cit. 03. 02. 2014).

[19] *Pricing.* TeamDrive Systems GmbH. URL: http://www.teamdrive.com/pricing.html (cit. 03. 02. 2014).

[20] *Summary of Features*. TeamDrive Systems GmbH. URL: http://www.teamdrive.com/feature_list.html (cit. 03. 02. 2014).

[21] *Free Choice of Server*. TeamDrive Systems GmbH. URL: http://www.teamdrive.com/free_server_selection.html (cit. 03. 02. 2014).

[22] *Cloud Storage Capacity (to share with unlimited users)*. TeamDrive Systems GmbH. URL: http://www.teamdrive.com/cloud_storage.html (cit. 03. 02. 2014).

[23] *TeamDrive Server*. TeamDrive Systems GmbH. URL: http://www.teamdrive.com/teamdrive_Server_en.html (cit. 03. 02. 2014).

[24] *TeamDrive Personal Server*. TeamDrive Systems GmbH. URL: http://www.teamdrive.com/teamdrive_personal_server.html (cit. 03. 02. 2014).

[25] *TeamDrive SecureOffice*. TeamDrive Systems GmbH. URL: http://www.teamdrive.com/secure_office_.html (cit. 03. 02. 2014).

[26] *Collaboration*. TeamDrive Systems GmbH. URL: http://www.teamdrive.com/collaboration.html (cit. 03. 02. 2014).

[27] *Security*. TeamDrive Systems GmbH. URL: http://www.teamdrive.com/security.html (cit. 03. 02. 2014).

[28] *Security and Architecture*. TeamDrive Systems GmbH. URL: http://www.teamdrive.com/security_and_architecture.html (cit. 03. 02. 2014).

[29] *Mega (service)*. URL: http://en.wikipedia.org/wiki/Mega_(website) (cit. 02. 02. 2014).

[30] *MEGA*. Mega, Ltd. URL: http://mega.co.nz (cit. 02. 02. 2014).

[31] Jakub Čížek. *Je tu Mega a chce rozcupovat Dropbox i klasická úložiště*. URL: http://www.zive.cz/clanky/je-tu-mega-a-chce-rozcupovat-dropbox-i-klasicka-uloziste/sc-3-a-167231/default.aspx (cit. 02. 02. 2014).

[32] Owen Williams. *Kim Dotcom's Mega to release Windows Phone app in early 2014*. URL: http://thenextweb.com/apps/2013/11/30/mega-release-windows-phone-client-early-2014/ (cit. 02. 02. 2014).

[33] Ian Paul a Pavel Kreuziger. *Mega: Zkušenosti se šifrovaným cloudem - 2. díl*. URL: http://pcworld.cz/internet/mega-zkusenosti-se-sifrovanym-cloudem-2-dil-45498 (cit. 02. 02. 2014).

[34] *Help Centre - Security & Privacy*. Mega, Ltd. URL: https://mega.co.nz/#help/security (cit. 02. 02. 2014).

[35] *Help Centre - Account*. Mega, Ltd. URL: https://mega.co.nz/#help/account (cit. 02. 02. 2014).

[36] *Kim Dotcom's NSA-proof messaging, video chat service due in 2014*. URL: http://bgr.com/2013/11/08/kim-dotcom-mega-messaging-service/ (cit. 02. 02. 2014).

[37] *BitTorrent Sync*. URL: http://en.wikipedia.org/wiki/BitTorrent_Sync (cit. 03. 02. 2014).

[38] Roman Bořánek. *BitTorrent Sync: úložiště bez pána*. URL: http://www.root.cz/clanky/bittorrent-sync-uloziste-bez-pana/ (cit. 03. 02. 2014).

[39] *Technology*. BitTorrent, Inc. URL: http://www.bittorrent.com/sync/technology (cit. 03. 02. 2014).

[40] Karel Kilián. *Bittorrent Sync: domácí variace na cloud šlape bez vzdálených serverů.* URL: http://www.cnews.cz/recenze/bittorrent-sync-domaci-variace-na-cloud-slape-bez-vzdalenych-serveru (cit. 03. 02. 2014).

[41] Jeff Preshing. *Hash Collision Probabilities.* URL: http://preshing.com/20110504/hash-collision-probabilities/ (cit. 28. 04. 2014).

[42] *SHA-2.* URL: http://en.wikipedia.org/wiki/SHA-2 (cit. 28. 04. 2014).

[43] *Extensible Markup Language.* URL: http://cs.wikipedia.org/wiki/Extensible_Markup_Language (cit. 28. 04. 2014).

[44] *JavaScript Object Notation.* URL: http://cs.wikipedia.org/wiki/JavaScript_Object_Notation (cit. 28. 04. 2014).

[45] *JSON vs. XML: Some hard numbers about verbosity.* URL: http://www.codeproject.com/Articles/604720/JSON-vs-XML-Some-hard-numbers-about-verbosity (cit. 28. 04. 2014).

[46] *The JSON Data Interchange Format.* Standard ECMA-404. Geneva, CH: ECMA International, říj. 2013. URL: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf (cit. 28. 04. 2014).

[47] Damien Giry. *Cryptographic Key Length Recommendation.* URL: http://www.keylength.com/en/4/ (cit. 28. 04. 2014).

[48] *Transport Layer Security.* URL: http://en.wikipedia.org/wiki/Transport_Layer_Security (cit. 28. 04. 2014).

[49] *Pretty Good Privacy.* URL: http://en.wikipedia.org/wiki/Pretty_Good_Privacy (cit. 28. 04. 2014).

[50] *Web of trust.* URL: http://en.wikipedia.org/wiki/Web_of_trust (cit. 28. 04. 2014).

[51] *libjson.* URL: http://libjson.sourceforge.net/ (cit. 30. 04. 2014).

[52] *jzon.* URL: https://code.google.com/p/jzon/ (cit. 30. 04. 2014).

[53] *JSON Spirit: A C++ JSON Parser/Generator Implemented with Boost Spirit.* URL: http://www.codeproject.com/Articles/20027/JSON-Spirit-A-C-JSON-Parser-Generator-Implemented (cit. 30. 04. 2014).

[54] *Jansson.* URL: http://www.digip.org/jansson/ (cit. 30. 04. 2014).

[55] *rapidjson.* URL: https://code.google.com/p/rapidjson/ (cit. 30. 04. 2014).

[56] *OpenSSL.* URL: https://www.openssl.org/ (cit. 30. 04. 2014).

[57] *The GnuTLS Transport Layer Security Library.* URL: http://www.gnutls.org/index.html (cit. 30. 04. 2014).

[58] *Implementace v GnuTLS.* URL: http://webcity.wz.cz/pd/clanek.php?cid=503 (cit. 30. 04. 2014).

[59] *GnuTLS 3.3.2.* URL: http://www.gnutls.org/manual/gnutls.html (cit. 30. 04. 2014).

[60] *How do I make a PGP key?* URL: http://askubuntu.com/questions/100281/how-do-i-make-a-pgp-key (cit. 30. 04. 2014).

[61] Martin Volf. *Distribuované úložiště dat.* Diplomová práce. Praha, leden 2014. URL: https://support.dce.felk.cvut.cz/mediawiki/images/6/69/Dp_2014_volf_martin.pdf (cit. 02. 05. 2014).

[62] Petr Tuček. *Distribuované úložiště dat*. Diplomová práce. Praha, červen 2012. URL: `https://dip.felk.cvut.cz/browse/pdfcache/tucekpe2_2012dipl.pdf` (cit. 02. 05. 2014).