Kauno technologijos universitetas

Informatikos fakultetas

Efektyvaus duomenų saugojimo paskirstytoje saugykloje metodo sukūrimas ir tyrimas

Baigiamasis magistro projektas

Matas Grišius

Projekto autorius

Doc. dr. Nerijus Morkevičius

Vadovas

Kaunas, 2023

Kauno technologijos universitetas

Informatikos fakultetas

Efektyvaus duomenų saugojimo paskirstytoje saugykloje metodo sukūrimas ir tyrimas

Baigiamasis magistro projektas

Informacijos ir informacinių technologijų sauga (6211BX008)

Matas Grišius

Projekto autorius

Doc. dr. Nerijus Morkevičius

Vadovas

Prof. Jevgenijus Toldinas

Recenzentas

Kaunas, 2023

Kauno technologijos universitetas

Informatikos fakultetas

Matas Grišius

Efektyvaus duomenų saugojimo paskirstytoje saugykloje metodo sukūrimas ir tyrimas

Akademinio sąžiningumo deklaracija

Patvirtinu, kad:

1. baigiamąjį projektą parengiau savarankiškai ir sąžiningai, nepažeisdama(s) kitų asmenų autoriaus ar kitų teisių, laikydamasi(s) Lietuvos Respublikos autorių teisių ir gretutinių teisių įstatymo nuostatų, Kauno technologijos universiteto (toliau – Universitetas) intelektinės nuosavybės valdymo ir perdavimo nuostatų bei Universiteto akademinės etikos kodekse nustatytų etikos reikalavimų;

2. baigiamajame projekte visi pateikti duomenys ir tyrimų rezultatai yra teisingi ir gauti teisėtai, nei viena šio projekto dalis nėra plagijuota nuo jokių spausdintinių ar elektroninių šaltinių, visos baigiamojo projekto tekste pateiktos citatos ir nuorodos yra nurodytos literatūros sąraše;

3. įstatymų nenumatytų piniginių sumų už baigiamąjį projektą ar jo dalis niekam nesu mokėjęs (-usi);

4. suprantu, kad išaiškėjus nesąžiningumo ar kitų asmenų teisių pažeidimo faktui, man bus taikomos akademinės nuobaudos pagal Universitete galiojančią tvarką ir būsiu pašalinta(s) iš Universiteto, o baigiamasis projektas gali būti pateiktas Akademinės etikos ir procedūrų kontrolieriaus tarnybai nagrinėjant galimą akademinės etikos pažeidimą.

Matas Grišius

Patvirtinta elektroniniu būdu

Grišius Matas. Efektyvaus duomenų saugojimo paskirstytoje saugykloje metodo sukūrimas ir tyrimas. Magistro baigiamasis projektas / vadovas doc. dr. Nerijus Morkevičius; Kauno technologijos universitetas, informatikos fakultetas.

Studijų kryptis ir sritis (studijų krypčių grupė): Informatikos inžinerija, informacijos ir informacinių technologijų sauga.

Reikšminiai žodžiai: saugykla, informacijos saugumas, paskirstyta saugykla.

Kaunas, 2023. 60 p.

Santrauka

Kompiuterių naudotojai dažnai susiduria su problema, kai dėl disko gedimų ar techninių klaidų praranda svarbius kietojo disko duomenis, ypač jei jie neturi atsarginės duomenų kopijos arba nesaugo jos atskirame diske. Vienas iš sprendimų yra tokias atsargines kopijas ar kitus svarbius failus saugiai užšifruotus laikyti kitų vartotojų kompiuteriuose, tokiu būdu užtikrinant aukštesnį duomenų pasiekiamumą bei vientisumą, tačiau neprarandant jų konfidencialumo. Siūloma sistema sukuria paskirstytą saugyklą, kuri informaciją saugo vartotojų asmeninių kompiuterių diskuose bei naudoja specialius klaidų taisymo kodus failams joje įrašyti. Sistemos vartotojai gali leisti sistemai naudotis jų disko vieta, o už tai mainais gauti vietos saugykloje, kuri pasižymi aukštesniu failų patikimumu. Šiame magistro baigiamajame projekte yra pristatomas paskirstytos saugyklos metodas bei atliekamas jo tyrimas. Darbas susideda iš keturių dalių.

Pirmoje dalyje pateikiama šiam metodui įgyvendinti reikalingų technologijų analizė ir panašių sprendimų apžvalga. Antroje dalyje pateikiama siūlomo sprendimo koncepcinė vizija ir veiklos planas. Trečioje dalyje šis metodas realizuojamas praktiškai ir detaliau aprašoma implementacija. Ketvirtajame skyriuje atliekamas metodo prototipo tyrimas ir įvertinamas tinkamumas tokiai problemai spręsti.

Grišius Matas. Research and Evaluation of the Method for Effective Distributed Data Storage. Master's Final Degree Project / supervisor doc. dr. Nerijus Morkevičius; Faculty of Informatics, Kaunas University of Technology.

Study field and area (study field group): Informatics engineering, information and information technology security.

Keywords: storage, information security, distributed storage.

Kaunas, 2023. 60 p.

Summary

Computer users often encounter the issue of losing crucial data on their hard disk due to disk failures or technical errors, especially when they have not backed up their data or stored it on a separate disk. This paper proposes a solution that involves securely encrypting backups and important files on other users' computers. This approach ensures higher availability and integrity of the data without compromising its confidentiality. The proposed system creates distributed storage on users' personal computer disks and uses special error-correcting codes to store files there. Users could trade their disk space for higher file reliability. The paper is divided into four parts.

The first part involves an analysis of the necessary technologies for implementing this approach and a review of similar solutions. The second part presents the conceptual vision and operational plan of the proposed solution. The third part focuses on the practical implementation, providing a detailed description of the process. Finally, the fourth chapter evaluates a prototype of the approach and assesses its suitability for addressing the aforementioned problem.

Turinys

Lentelių sąrašas

Paveikslų sąrašas

Santrumpų ir terminų sąrašas

Santrumpos:

HDD (angl. *hard disk drive*) – kietasis diskas, skirtas duomenims saugoti.

SSD (angl. *solid-state drive)* – duomenims saugoti skirtas diskas, grįstas puslaidininkiais.

RAID (angl. *redundant array of independent disks*) – technologija, leidžianti apjungti keletą fizinių diskų į bendrą duomenų saugyklą, tokiu būdu užtikrinant aukštesnį pasiekiamumą ir/arba efektyvumą.

IETF (angl. *Internet Engineering Task Force*) – savanoriška organizacija, kurianti interneto standartus bei kitas technines specifikacijas.

CIA (angl. *confidentiality, integrity, availability*) – konfidencialumas, pasiekiamumas bei vientisumas – trys pagrindiniai saugios duomenų saugyklos komponentai.

NIST (angl. *National Institute of Standards and Technology*) – nacionalinė Jungtinių Amerikos Valstijų agentūra, kurios pagrindinis tikslas yra skatinti inovacijas ir technologijų naujoves.

DB (angl. *database*) – duomenų bazė.

Terminai:

Paskirstyta saugykla – informacijos saugojimo sistema, leidžianti duomenis laikyti atskiruose fiziniuose diskuose ar serveriuose.

Įvadas

Darbo problematika ir aktualumas

Beveik kiekviena kompiuterinė programa, kiekvienas žiniatinklio puslapis ar apskritai bet kuri kompiuterinė sistema yra priversta saugoti tam tikrus duomenis. Saugoma informacija gali būti pati įvairiausia: tekstiniai failai, nuotraukos, vaizdo įrašai, dokumentai, žemėlapiai, įvairūs nustatymų failai, metaduomenys ir begalė kitokių tipų. Dauguma programų net negalėtų atlikti savo funkcijų, jei neturėtų galimybės išsaugoti ir vėliau perskaityti ir atgauti išsaugotą informaciją.

Nauji išradimai, tobulėjanti pramonė generuoja vis didesnius informacijos kiekius. Pavyzdžiui, tobulėjant fotokameroms, gerėja jų nuotraukų kokybė, taškų kiekis, kartu ir užimama vieta, o visas sukurtas nuotraukas reikia išsaugoti. Pramonė tobulėja labai sparčiai, kiekvienais metais išleidžiami nauji tinklų standartai, naujos mobiliojo ryšio technologijos, pažangesni *Bluetooth* standartai, o visų jų privalumas dažniausiai yra didesnė sparta. Kompiuterių gamyba taip pat nestovi vietoje, kuriami naujos kartos procesoriai, gebantys apdoroti vis didesnius informacijos kiekius. Visa tai lemia, kad informacijos kiekiai ateityje dar labiau augs, o informaciją reikės vienokiu ar kitokiu būdu saugoti.

Vis didesniems informacijos kiekiams saugoti yra pasiruošę ir diskų gamintojai: kuriami vis didesnės talpos kietieji (HDD) ar puslaidininkiais grįsti (SSD) diskai, jų gamybos kaina nuolatos mažėja, tačiau tai iki galo neišsprendžia vienos iš pagrindinių informacijos saugojimo problemų – informacijos praradimo. Įvykus disko klaidai, visa jame saugoma informacija gali būti prarasta. Tai ypač aktualu naujos kartos SSD diskams, po kurių gedimo informacijos atkurti yra beveik neįmanoma. Taigi, reikalingas sprendimas, kuris leistų ne tik saugoti informaciją, bet kartu ir užtikrinti, kad ji nebus prarasta laikui bėgant dėl disko klaidos.

Tarkime, kad kompiuterio naudotojas turi vieną didelės talpos duomenų diską. Jis gali saugoti didelį informacijos kiekį, tačiau yra didelė tikimybė ją visą prarasti. Dažnu atveju naudotojas labiau vertintų duomenų saugojimo būdą, užtikrinantį aukštesnę apsaugą nuo visiško praradimo, mainais už mažesnę disko talpą. Tą galima pasiekti, jei tokie vartotojai jungtųsi į grupes ir leistų pasinaudoti vienas kito duomenų diskais.

Įvykus klaidai viename diske, informacija nebus visiškai prarasta, jei ji buvo saugoma išskirstytu būdu keliuose diskuose. Vienas iš tokio saugojimo pavyzdžių yra RAID 1 jungtiniai diskai, kur yra kuriamas dublikatas ir informacija yra saugoma dviejuose diskuose, todėl net ir praradus vieną diską, galima visiškai atkurti originalią informaciją iš kito disko. Tiesa, tokiu saugojimo būdu failui išsaugoti sunaudojama du kartus daugiau vietos, negu įprastai, todėl vargu, ar tai yra pats efektyviausias būdas.

Šio darbo tikslas yra sukurti efektyvų duomenų saugojimui paskirstytoje saugykloje metodą, kuris geba užtikrinti duomenų konfidencialumą, pasiekiamumą, bei vientisumą, taip pat identifikuoti tokio saugojimo būdo iššūkius, pasiūlyti savo realizaciją, kaip efektyviai saugoti informaciją paskirstytoje saugykloje ir atlikti tokios saugyklos kiekybinį ir kokybinį vertinimą.

Darbo tikslas ir uždaviniai

Tikslas: sukurti efektyvų duomenų saugojimo metodą paskirstytoje saugykloje ir atlikti jo tyrimą.

Uždaviniai:

1. Išgryninti pagrindines duomenų saugojimo paskirstytoje saugykloje problemas.
2. Išanalizuoti esamus duomenų saugojimo metodus paskirstytoje saugykloje.
3. Pasiūlyti savo sprendimą duomenims saugoti paskirstytoje saugykloje.
4. Praktiškai realizuoti pasiūlyto sprendimo metodo prototipą.
5. Eksperimentiškai ištirti šio metodo veikimą bei atlikti kiekybinę analizę.
6. Atlikti pasiūlyto metodo prototipo kokybinę analizę.

Darbo struktūra

Darbas sudarytas iš keturių dalių. Duomenų saugojimo paskirstytoje saugykloje analizės dalyje išanalizuoti egzistuojantys sprendimai duomenų saugojimui, taip pat supažindinama su pagrindinėmis duomenų saugojimo paskirstytoje saugykloje problemomis bei kiekvienai iš jų pasirenkamas sprendimas. Antroje dalyje pristatomas saugyklos metodas ir koncepcinė vizija, aprašomas projektuojamos sistemos veikimo principas. Trečioje dalyje aprašoma sprendimo prototipo realizacija ir platesnės sistemos įgyvendinimo detalės, o ketvirtoje dalyje atliekamas bei pateikiamas sukurto metodo kiekybinis bei kokybinis tyrimas prie tam tikrų apkrovų.

1. Duomenų saugojimo paskirstytoje saugykloje analizė

1.1. Paskirstyta duomenų saugykla

Paskirstytų saugyklų paskirtis yra sukurti patikimą būdą laikyti informacijai ilgą laiką. Tokia saugykla susideda iš didelio kiekio nepriklausomų serverių, dar vadinamų taškais arba mazgais, kurie individualiai gali būti ir nepatikimi, tačiau jų visuma sudaro patikimą terpę duomenims saugoti.

Patikimumas tokiose saugyklose užtikrinamas pertekliškumu. Pats paprasčiausias duomenų pertekliškumo pavyzdys – dubliavimas, kuris taip pat neretai naudojamas duomenų saugojimo sistemose. Jei iš failo sukursime antrą kopiją ir patalpinsime ją į kitą serverį, tuomet net ir vienam iš serverių sugedus vis dar galėsime atgauti savo įkeltą failą iš veikiančio serverio. Visgi, bendru atveju toks duomenų dubliavimas reikalauja didelio kiekio pertekliškumo, o tokį patį patikimumo lygį galima užtikrinti ir efektyvesniais metodais, pavyzdžiui, informacijos saugojimas naudojantis klaidų taisymo kodais.

Klaidų taisymo kodai paskirsto failą į didelį kiekį dalių, vadinamų fragmentais. Kiekvienas fragmentas saugomas vis kitame serveryje, todėl net ir sugedus daliai iš serverių, kitą failo dalį vis dar galima atstatyti. Taip pat, klaidų taisymo kodai saugo papildomus duomenis (metaduomenis) apie sudarytus fragmentus bei apie jų visumą, pagal kurią, reikalui esant, tam tikromis situacijomis galima atkurti net ir prarastus failų fragmentus ir tokiu būdu atgauti pilną failo informaciją. Žinoma, šie papildomi metaduomenys užima papildomos vietos, tačiau ši papildomai užimama vieta nepalyginamai mažesnė už pilno failo dublikatą.

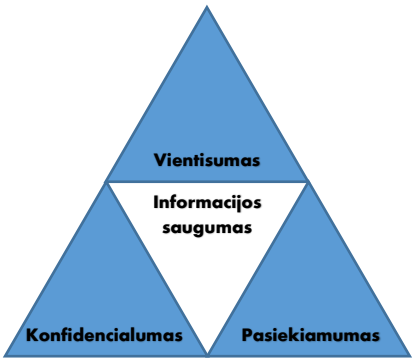
Nors toks būdas saugoti failus paskirstytoje saugykloje ir yra efektyvus, tačiau jis įneša ir tam tikrą kompleksiškumo lygį. Vietoj to, kad saugykla saugotų pilnus failus, reikia saugoti jų fragmentus, nesumaišyti skirtingų failų fragmentų tarpusavyje, fragmentus saugoti skirtinguose serveriuose ir reguliariai tikrinti, apie nutolusių mazgų būseną. Taip pat, norint vėliau atgauti pilną failą, reikia surinkti fragmentus iš nutolusių serverių, juos sujungti į bendrą failą ir tik tada pateikti galutiniam vartotojui. Taip pat negalima pamiršti saugumo aspekto – kadangi failų fragmentai dažnai keliauja tinklu iš vieno serverio į kitą, jie turi padidintą riziką būti sukompromituoti, kuria būtina pasirūpinti.

Kiekviena paskirstyta duomenų saugykla naudoja skirtingus metodus bei algoritmus savo veikimui užtikrinti. Nuo naudojamų įrankių ar algoritmų priklauso saugyklos saugumas bei bendras saugyklos efektyvumas. Saugyklos efektyvumas gali būti išreiškiamas įvairiomis metrikomis:

- vietos panaudojimo efektyvumas – nurodo, kokia dalis saugomų duomenų yra pertekliniai;
- greitaveika – nurodo kaip greitai vyksta duomenų įkėlimas į saugyklą bei duomenų parsisiuntimas iš saugyklos;
- vidinis sistemos našumas –naudojamų algoritmų našumas apdorojant failus;
- patikimumas – nurodo sistemos patikimumo priklausomybę nuo pasiekiamų bei nepasiekiamų mazgų santykio;
- energijos efektyvumas – nurodo, kiek elektros energijos sunaudojama mazgų darbui palaikyti;
- tinklo apkrova – parodo kaip stipriai yra apkraunamas tinklas dalinantis informacija tarp mazgų;

1.2. Informacijos saugumo CIA trikampis

CIA – (angl. CIA – *Confidentiality, Integrity, Availability*) – tai sąvoka, apibrėžianti tris fundamentalius duomenų laikymo kriterijus, kuriuos būtina užtikrinti kiekvienai sistemai: konfidencialumą, vientisumą bei pasiekiamumą. Tai yra pagrindiniai kriterijai, pagal kuriuos galima projektuoti bei vertinti duomenų saugojimo sistemas (**1.1 pav.**).



1.1 pav. CIA trikampis – trys esminės informacijos saugumo savybės

1.2.1. Konfidencialumas

Konfidencialumas nurodo, kiek pastangų yra skiriama tam, kad saugoma informacija išliktų privati ir saugojama nuo nesusijusių vartotojų. Praktikoje dažniausia šią savybę įgyvendina kontrolė, kas gali pasiekti duomenis. Tai reiškia, kad tik autorizuoti vartotojai turi prieigą prie tam tikrų duomenų, o tie, kurie tos prieigos neturi, privalo būti blokuojami nuo informacijos pasisavinimo. Pavyzdžiui, informacija apie banko klientus turėtų būti pasiekiama tik paties banko darbuotojams, ir tik tiems, kurie aptarnauja klientus. Tokia informacija turėtų būti saugoma ne tik nuo visų kitų banko darbuotojų, bet ir tuo jos turėtų būti neįmanoma pasiekti pašaliniams žmonėms [1].

Pažeidimai duomenų konfidencialumui gali būti sukeliami pačiais įvairiausiais būdais, pavyzdžiui, tiesiogine ataka, siekiant gauti neautorizuotą prieigą prie sistemos, programų, o kartu ir duomenų bazės, siekiant pavogti duomenis arba juos pakeisti. Kitas galimas būdas yra tinklo žinučių perėmimas, tinklo skenavimas, „žmogaus viduryje“ ataka, kai duomenys gaunami ne tiesiai iš duomenų bazės, bet jų keliavimo tinklu metu. Dar vienas būdas pažeisti duomenų konfidencialumą yra socialinės atakos bei socialinė inžinerija, kai žmogus gali apsimesti, tarkime, banko darbuotoju ir paskambinęs aukai prašyti jam pateikti slaptą informaciją. Dažnai informacijos konfidencialumas yra pažeidžiamas ir dėl žmogiškosios klaidos, neatsargumo ar nepakankamo rūpinimosi duomenimis, pavyzdžiui, programuotojas nenaudoja slaptažodžių šifravimo kriptografiniais algoritmais arba tinklo administratorius neužtikrina vidinio tinklo paskyrų teisių kontrolės. Yra atveju, kai informacija sukompromituojama net ir fiziniais metodais, tarkime, piktadariui ją matant nepastebimai aukos monitoriuje. Dar vienas dažnas būdas pavogti duomenis yra netinkamai įgyvendinta autentifikacija, kurią piktavalis gali apeiti ir pasiekti sau nepriklausančią informaciją.

Duomenų konfidencialumą galima užtikrinti įvairiais metodais, pavyzdžiui, skirstant duomenis į kritiškai slaptus bei ne tokius slaptus, kartu užtikrinant adekvačią apsaugą, kiekvienam iš šių tipų. Slapti duomenis taip pat turėtų būti pasiekiami tik įsitikinus, kad vartotojų autentifikacija yra tinkamai įgyvendinta ir vartotojas yra prisijungęs. Papildomai turėtų būti naudojamos kriptografinės funkcijos, kurios leistų saugoti tik užšifruotus duomenis, kad piktavalis net ir gavęs prie jų prieigą, negalėtų jais

pasinaudoti ir tik autorizuotas vartotojas, turintis susietą raktą, gali juos iššifruoti. Ne mažiau svarbus dalykas yra ir kiekvieno vartotojo mokymas, kaip elgtis su informacija ir kaip ją apsaugoti, kaip užtikrinti savo paskyros slaptažodžio bei apskritai savo tapatybės apsaugą ir t. t.

1.2.2. Vientisumas

Jei pažiūrėtume į būdvardžio „integralus“ sąvoką žodyne, tai šio žodžio sinonimai būtų „visas“, „užbaigtas“ arba „pilnas“. Informacinėje aplinkoje vientisumas, kalbant apie duomenis, reiškia, kad šie duomenys nebuvo pakeisti, tai yra originali, autentiška ir patikima jų versija, todėl jais tikrai galima pasitikėti. Tarkime, jei perkame ką nors elektroninėje parduotuvėje, tai mes tikimės, kad nuo įdėjimo į krepšelį momento, iki mokėjimo bankiniu pavedimu momento prekės kaina, tipas bei išvaizda nebus pakitusi ir tai bus vis dar ta pati prekė, su ta pačia tikima kaina bei kitomis ypatybėmis [1].

Vientisumas dažnai naudojamas kartu užtikrinti ne tik pačio objekto vidinę informaciją, tačiau ir objekto autentiškumą bei neišsiginamumą, taip sujungiant informacijos vienetą kartu su jo autoriumi. Neišsiginamumas užtikrina, kad duotą informacijos vienetą sukūrė, modifikavo ir kartu už jį yra atsakingas būtent nurodytos tapatybės asmuo. Svarbu paminėti, kad nors autentiškumas ir neišsiginamumas yra labai panašūs terminai, kalbant apie saugumą, autentiškumas pats iš savęs neužtikrina neišsiginamumo, tačiau neišsiginamumas kartu garantuoja ir duomenų autentiškumą. Taip yra todėl, kad duomenų neišsiginamumas užtikrina transakciją ir panaikina abejonę, kas yra už ją atsakingas, todėl jei autorius yra aiškiai žinomas, galima nustatyti, kad prieš tai buvo atlikta autentifikacija. Tarkime, jei vartotojas siunčia elektroninį laišką, kurį jis pasirašo elektroniniu parašu, tai šis parašas bus ne tik įrodymas, kad žinutė pas gavėją atėjo būtent tokia, kokią ir išsiuntė siuntėjas, tačiau ir įrodymas, kad ši žinutė buvo parašyta ir atsiųsta būtent siuntėjo, o ne jokio kito asmens.

Duomenų vientisumas taip pat gali būti pažeistas. Integralumą gali pažeisti tiek tiesioginė ataka (neteisėtas failo ar duomens pakeitimas, sisteminių užrašų slėpimas, ištrynimas, vientisumo sunaikinimas), tiek ir netyčiniai vektoriai, tokie kaip žmogiškoji klaida, nepakankamas rūpestis, programavimo klaidos, netinkama konfigūracija ar tiesiog dalies duomenų praradimas dėl fizinio laikmenos pažeidimo.

Duomenų vientisumą galima užtikrinti naudojant kriptografinius šifrus, maišos funkcijas, elektroninį parašą ar skaitmeninius sertifikatus, versijų kontrolę, auditą bei prieigos valdymą.

1.2.3. Pasiekiamumas

Informacijos apsaugos srityje terminas „pasiekiamumas“ gali būti aiškinamas kaip „patikimama duomenų prieiga reikiamu metu“. Sistemos, programos ar duomenys būtų beverčiai, jeigu naudotojai negalėtų jų pasiekti tada, kai jiems jų reikia [1].

Sistemos pasiekiamumas gali nukentėti dėl labai daug priežasčių, kurios gali būti susiję tiek su aparatinės, tiek ir su programinės įrangos gedimais. Pasiekiamumas gali nukentėti ir dėl oro sąlygų, stichinių nelaimių, elektros energijos tiekimo sutrikimų, žemės drebėjimų ir t. t. Dažnai sistema gali tapti nepasiekiama ir dėl žmogiškųjų klaidų, pavyzdžiui, sistemos operatoriaus / administratoriaus klaidos. Žinoma, ši savybė taip pat neatsiejama ir nuo programišių atakų: pasiekiamumą gali sutrikdyti ir bene plačiausiai žinoma atakos rūšis – paslaugos atsisakymo (angl. *DoS*) ataka, kurios

metu dėl didelio kiekio atsiųstų užklausų sistema kartais ne tik labai sulėtėja, tačiau kartais ir visiškai nutraukia darbą ir tampa nebepasiekiama.

Pasiekiamumas dažniausia užtikrinamas įrangos bei procesų (serverių, tinklų, programų bei paslaugų) pertekliškumu, taip pat naudojant aparatinę ar programinę įrangą, kuri turi numatytas taisykles, kaip elgtis klaidos atveju, vietoj to, kad visiškai nutraukti darbą. Taip pat visos naudojamos programos, bibliotekos, įskiepiai bei kita programinė įranga privalo būti nuolat atnaujinama į naujausias versijas, o aparatinė įranga taip pat gerinama, plečiama bei naujinama atsižvelgiant į naudojimo srautą. Serverių patalpos bei patys serveriai turėtų būti paruošti naudotis alternatyviu energijos šaltiniu, esant įprastam elektros energijos deficitui. Programos ir serveriai privalo naudoti taisykles, kurios bandytų tikrinti klientus, IP adresus bei kitą informaciją, kuri leistų apsisaugoti nuo paslaugos atsisakymo atakos.

1.3. Klaidų taisymo kodai

Klaidų taisymo kodai yra naudojami pataisyti duomenis, kurie buvo siunčiami per nepatikimą kanalą, kuriame galimai galėjo būti triukšmų ar kitokių signalo iškraipymų. Įprastai po duomenų siuntimo tokiu kanalu prasideda antra fazė, kurios metu gavėjas siunčia žinutę siuntėjui apie tai, kuriuos paketus reikėtų pakartoti. Naudojant klaidų taisymo kodus nereikalingas atskiras kanalas gavėjui, nes siuntėjas dar prieš siųsdamas prie failo prideda perteklinės informacijos, pavyzdžiui, kontrolines sumas, kurios leidžia gavėjui net ir gavus ne visus paketus atkurti failą į originalią jo versiją ir nereikia prašyti siuntėjo pakartoti siuntimo procesą.

1.3.1. Fontaniniai kodai

Fontaniniai kodai – tokie kodai, kurie geba pagal duotą failą generuoti paketus tokiu būdu, kad paketų kiekis gali būti laisvai pasirenkamas ir netgi begalinis, o vėliau norint iš paketų atstatyti pradinį failą, pakanka turėti bet kokį rinkinį paketų, kurių bendras dydis yra vos didesnis už pradinio failo dydį. Kad būtų lengviau įsivaizduoti, galime teigti, kad iš failo yra sukuriamas fontanas, kuris amžinai ir nuolatos tiekia naujus paketus. Jei įsivaizduosime, kad turime kibirą, kurį nunešame ir pastatome šalia fontano, tai vienintelė sąlyga, kad būtų įmanoma atstatyti pradinį failą, yra ta, kad kibire turi „prilyti“ toks pat kiekis paketų, koks ir yra pradinio failo dydis.

Išskirtinė fontaninių kodų savybė yra ta, kad iš pradinio failo galima sugeneruoti potencialiai begalinį kiekį paketų, o pats generuojamas kiekis gali būti lengvai nustatomas bet kurią akimirką. Galima sakyti, kad fontaniniai kodai yra universalūs ta prasme, kad jų generuojami paketai gali tikti iškart dideliam kiekiui klausytojų, nes kiekvienas paketas tinka kiekvienam iš jų, skirtingai nei įprastoje situacijoje, kai failas tiesiog suskaidomas į dalis ir tada kiekvienam iš klausytojų iki pilno failo gali trūkti vis kitos failo dalies, o pasikartojančios dalys yra bevertės. Fontaniniai kodai taip pat pasižymi nedideliu sudėtingumu šifruojant ir iššifruojant.

1.3.2. *Luby Transform* kodai

Luby transformacijos (LT) kodai yra klaidų taisymo kodų klasė, kurią 2002 m. pristatė Michael Luby. Šie kodai skirti efektyviai atkurti prarastus ar sugadintus duomenis siuntimo paketais sistemose. Jie ypač gerai veikia tokiose scenarijuose, kai dažniau paketas tampa visai nepasiekiamas, negu tampa sugadintas. LT kodai plačiai naudojami tokiose srityse kaip vaizdo transliavimas ar failų perdavimas.

Luby Transform veikimo (kodavimo) metu pagal pradinius duomenis sukurtiems simboliams (paketams) yra pritaikomos atsitiktinės tiesinės transformacijos, tokiu būdu sukuriant pradinių bei papildomų simbolių (paketų) rinkinį, vadinamą LT simboliais. Šie simboliai yra perduodami tinklu, o gavėjas surenka tam tikrą LT simbolių dalį. Pasinaudodamas LT kodų savybėmis, gavėjas gali sėkmingai iškoduoti pradinius duomenis spręsdamas tiesinių lygčių sistemą.

Vienas iš pagrindinių *Luby Transform* kodų privalumų yra tai, kad jie gali būti pritaikomi įvairioms tinklo sąlygoms. LT kodai gali pasiekti bet kokį klaidų taisymo lygį su didele sėkmingo iškodavimo tikimybe, reguliuojant perduodamų papildomų simbolių (paketų) skaičių. Dėl tokio lankstumo LT kodai gali veikti skirtingomis kanalo sąlygomis, kartais net ir prisitaikyti prie dinamiškų tinklo pokyčių. Be to, LT kodai yra beveik idealūs dėl minimalios informacijos poreikio, siekiant sėkmingai iškoduoti pradinį failą [2].

1.3.3. *Raptor* kodai

Raptor kodai iš esmės yra patobulinti *Luby Transform* kodai. *Raptor* kodai yra pats efektyviausias fontaninis kodas, kuris gali būti vadinamas beveik idealiu. Idealus fontaninis kodas yra toks, kai pradinio failo atkūrimui pakanka lygiai tokio paties kiekio paketų, kaip ir pradinio failo užimamas dydis. *Raptor* kodas taip pat pasižymi išskirtiniu savo kodavimo ir iškodavimo paprastumu. Pagal IETF (angl. *Internet Engineering Task Force*) šiuo metu yra standartizuoti du *Raptor* kodų pritaikymo variantai: *R10* ir *RaptorQ*. *R10* atsirado anksčiau ir dabar gali būti randamas įvairiuose technologijų standartuose, tokiuose kaip duomenų persiuntimas mobiliaisiais tinklais, palydovo komunikacijos, internetinė televizija ar vaizdo įrašų transliavimas [3].

Iš esmės *Raptor* kodo sukūrimui didelę reikšmę davė *Luby Transformation* kodai. *Raptor* kodai yra patobulinta jų versija, minimizuojant simbolius, kurių šifravimo bei iššifravimo trukmė yra aukštesnė nei linijinė ir stengiantis kuo daugiau simbolių sugeneruoti tokių, kuriuos būtų efektyvu tiek užšifruoti, tiek vėliau iššifruoti. Tokiu būdu buvo galima pranokti LT kodus efektyvumu.

RaptorQ yra pati naujausia *Raptor* kodų panaudojimo versija, kuri pasižymi padidintu patikimumu kartu išlaikant efektyvų užkodavimą ir iškodavimą. Dėl šios priežasties tikėtina, kad ateityje *RaptorQ* turėtų pakeisti R10 standartą. Šiuo metu *RaptorQ* kodas taip pat yra standartizuotas ir pagal RFC 6330 aprašą, ir pagal dabartines tendencijas tikimasi, kad šį kodą turėtų standartizuoti ir daugiau organizacijų. Šį kodavimą galima naudoti plataus spektro programose ir paskirtyse, pavyzdžiui kariuomenėje, esant poreikiui perduoti kritinių operacijų duomenis nestabilioje aplinkoje, kurioje prasta infrastruktūra ir gali atsirasti daug trikdžių [4].

1.3.3.1. *Qualcomm* įrankis, pagrįstas *RaptorQ* kodavimu.

Qualcomm sukurto įrankio *RaptorQ* [5] veikimo principas:

- Siuntėjo programa, naudodama *RaptorQ* koduotoją, sugeneruoja koduotą informaciją pagal duotą originalią informaciją.
- Koduota informacija yra siunčiama tinklu gavėjui.
- Dalis informacijos siuntimo metu yra prarandama ir nepasiekia gavėjo.
- Gavėjas iškoduoja informaciją, naudodamas *RaportQ* koduotoją. Jei gavėją pasiekia pakankamas kiekis duomenų (nepriklausomai kiek ar kurie paketai buvo prarasti), iškodavimas veikia be klaidų ir originalus failas yra atstatomas.

RaptorQ įrangos koduotojo bei iškoduotojo programinės dalys pasižymi šiomis savybėmis:

- Išskirtinai greitas kodavimas bei iškodavimas – efektyvumo skalėje tiek kodavimas, tiek iškodavimas yra linijinio greičio.
- Išskirtinė apsisaugojimo nuo praradimo ypatybė – sėkmingai iškoduoti galima bet kokį failą, jei buvo gautas pakankamas kiekis duomenų (praktiškai tai yra toks kiekis, kokio dydžio buvo ir pradinis failas), nepriklausomai nuo to kiek paketų buvo prarasta ar kokie paketai buvo prarasti.
- Lankstumas generuojant paketus pagal duotą failą – fontano principas pagal duotąjį failą leidžia laisvai pasirinkti generuojamų paketų kiekį. Šis kiekis gali būti praktiškai neribotas, o kiekvienas šių paketų bus naudingas ir gavėjui suteiks vis daugiau šansų atkurti pradinį failą.

Prieš siuntimą *RaptorQ* koduoja nurodytą failą, kuris programoje dar vadinamas pradiniu bloku, į daug vienodo dydžio fragmentų, kurie dar vadinami pradiniais simboliais. Pradinio bloko dydis gali būti konfigūruojamas. Tada *RaptorQ* koduotojas papildomai sugeneruoja taisymo simbolius, kurie yra tokio paties dydžio kaip ir pradiniai simboliai (kurie buvo sugeneruoti pagal pradinį bloką). Vėliau tinklu siunčiami koduoti simboliai, kurie yra pradinių bei taisymo simbolių deriniai.

Paprastai kiekvienas siunčiamas simbolis turi 32 bitų antraštę, kuri leidžia identifikuoti siunčiamą simbolį.

RaptorQ programinė įranga geba pradinį bloką išskaidyti į 1 – 56304 pradinius simbolius, o taisymo simbolių kiekis yra praktiškai neribotas ir tikrai pakankamas bet kokio tipo aplikacijai. Programa geba sukurti iki 2^{24} taisymo simbolių vienam pradiniui blokui.

Tikimybė sėkmingai atstatyti failą lyginant sėkmingai atsiųstų simbolių kiekį su pradinio failo simboliu kiekiu yra labai aukšta: Jei failas buvo padalintas į K kiekį simbolių, tuomet *RaptorQ* tikimybė sėkmingai atstatyti failą yra:

- 99% kai gauta K koduotų simbolių;
- 99.99% kai gauta K+1 koduotų simbolių;
- 99.9999% kai gauta K+2 koduotų simbolių.

Šios tikimybės galioja su bet kokio dydžio failu, bet kokiu kiekiu sugeneruotų simbolių pagal bloką, taip pat nepriklausomai nuo prarastų simbolių kiekio ar prarastų simbolių išsidėstymo. Tai reiškia, kad tikimybė priklausys tik nuo sėkmingai gautų simbolių kiekio net ir tada, kai bus prarasta tarkime 20% visų siųstų simbolių arba nors ir 90% simbolių arba, tarkime, bus prarasti visi siuntimo pradžioje siųsti simboliai, o sėkmingai išsisiųs tik tie, kurie buvo siunčiami transmisijos gale.

1.3.4. *Reed-Solomon* kodas

Reed-Solomon (RS) kodai yra klaidų taisymo kodų klasė, plačiai naudojama įvairiose skaitmeninio ryšio sistemose ir saugojimo įrenginiuose. Šiuos kodus 1960 m. sukūrė Irving S. Reed ir Gustav Solomon, ir nuo to laiko jie taikomi įvairiose srityse, pavyzdžiui, duomenų perdavimui palydoviniu ryšiu, informacijos saugojimui diskuose bei duomenų perdavimui nepatikimais (triukšmingais) kanalais.

Vienas iš pagrindinių *Reed-Solomon* kodų privalumų yra jų gebėjimas ištaisyti tiek atsitiktines, tiek ir vienoje vietoje susikaupusias klaidas. Į pradinius duomenis įtraukdami perteklinę informaciją, RS

kodai gali aptikti ir ištaisyti klaidas net tada, kai pažeista nemažai bitų. Dėl šios savybės šie kodai ypač tinkami naudoti tais atvejais, kai duomenų vientisumas yra labai svarbus.

Reed-Solomon kodai grindžiami polinomine aritmetika baigtiniame lauke. Kodavimo procesas apima duomenų naudojimą kaip polinomo koeficientą, o tada generuojami papildomi simboliai (paketai), perskaičiuojant polinominę lygtį skirtinguose taškuose. Šie papildomi simboliai pridedami prie pradinių duomenų ir sudaro koduotą pranešimą. Iškodavimas atliekamas naudojant tas pačias lauko savybes bei interpoliacijos metodus. Visa tai atlikus, gauname pradinius duomenis.

Reed-Solomon kodų privalumas yra tai, kad jie geba suderinti klaidų taisymo galimybę bei efektyvumą. Vartotojas gali reguliuoti kodo parametrus, pavyzdžiui, kodo ilgį ir papildomų simbolių skaičių, kad atitiktų konkrečius reikalavimus. Ilgesni kodai su daugiau pariteto simbolių užtikrina geresnį klaidų taisymą, tačiau reikalauja daugiau skaičiavimo išteklių. Ir atvirkščiai, trumpesni kodai yra efektyvesni, tačiau jų klaidų taisymo galimybės yra ribotos. Dėl tokio lankstumo RS kodus galima pritaikyti įvairioms taikomosioms programoms, atsižvelgiant į konkrečius jų poreikius [6].

1.3.5. BCH kodai

BCH (*Bose-Chaudhuri-Hocquenghem*) kodai yra klaidų taisymo kodų klasė, plačiai naudojama įvairiose srityse, įskaitant telekomunikacijas, duomenų saugojimą ir palydovinį ryšį. 1959 m. matematikų Raj Bose, D. K. Ray-Chaudhuri ir A. Hocquenghem sukurti BCH kodai pasižymi gebėjimu aptikti ir ištaisyti klaidas, atsirandančias perduodant ar saugant duomenis.

BCH kodai priklauso ciklinių klaidų taisymo kodų šeimai, o tai reiškia, kad jie turi savybę cikliškai keisti kodinio žodžio bitus, dėl to gaunamas kitas galiojantis kodinis žodis. Ši savybė ypač naudinga klaidų taisymui, nes leidžia atlikti efektyvias kodavimo ir dekodavimo operacijas.

Vienas iš pagrindinių BCH kodų privalumų yra jų gebėjimas aptikti ir ištaisyti kelias klaidas kodiniame žodyje. Projektuojant BCH kodus reikia kruopščiai parinkti kodo parametrus, tokius kaip kodo ilgis, pranešimo ilgis ir klaidų taisymo gebėjimas, atsižvelgiant į konkrečius taikymo srities reikalavimus. BCH kodų klaidų taisymo gebą galima reguliuoti keičiant šiuos parametrus, todėl galima rasti balansą tarp klaidų taisymo gebėjimo ir kodo efektyvumo.

Klaidoms aptikti ir ištaisyti BCH kodai naudoja algebrinius metodus. Jie remiasi baigtiniais laukais, kuriuose yra apibrėžiamos matematinės operacijos, primenančios gerai žinomas aritmetines operacijas. Naudodami polinominę aritmetiką baigtiniuose laukuose, BCH kodai gali aptikti ir ištaisyti kodiniame žodyje esančias klaidas. Kodavimo procese žinutės polinomas dalijamas iš generatoriaus polinomo, kad būtų gautas kodinis žodis, o dekodavimo procese klaidoms ištaisyti naudojami tokie algoritmai kaip *Berlekamp-Massey* algoritmas arba *Petersono-Gorenšteino-Zierlerio* algoritmas.

Praktiškai BCH kodai yra naudojami ten, kur duomenų vientisumas ir klaidų taisymas yra labai svarbūs. Jie naudojami klaidų taisymo algoritmuose, skirtuose duomenų saugojimo įrenginiams, pavyzdžiui, kietiesiems diskams ir atmintinėms, užtikrinant patikimą ir tikslų duomenų atkūrimą. Be to, BCH kodai taikomi ryšių sistemose, kur jie minimizuoti praradimus, atsiradusius dėl triukšmų ir kanalo trikdžių, taip padidindami bendrą kanalo patikimumą [7].

1.4. Failų šifravimas

Failų šifravimas yra vienas iš būdų užtikrinti duomenų konfidencialumą. Duomenų bazėje turėtų būti saugomi užšifruoti failai, kad piktavalis net ir neteisėtai gavęs priėjimą prie duomenų, negalėtu jais pasinaudoti. Kiekvienas šifras turi savo algoritmą, kuriam reikalingas raktas. Raktas gali būti skaičius, žodis ar frazė. Algoritmas naudojasi duotu raktu ir užšifruoja pasirinktą failą į nepaaiškinamą simbolių seką. Vėliau algoritmas geba pagal duotą raktą vėl iššifruoti simbolių seką į originalų failą.

Kriptografiniai algoritmai skirstomi į du potipius: simetriniai ir asimetriniai. Simetriniai šifrai nuo asimetrinių skiriasi tuo, kad juose tiek užšifravimui, tiek iššifravimui naudojamas lygiai tas pats raktas. Asimetrinių šifrų atveju raktai yra du, vienas skirtas užšifruoti informaciją, o su kitu ją galima iššifruoti ir atvirkščiai. Simetriniai šifrai yra labai sena technika, o asimetriniai šifrai yra naujesni. Įprastai pagrindinė simetrinio šifravimo problema yra rakto dalinimasis. Kadangi tas pats raktas gali tiek užšifruoti, tiek iššifruoti pranešimus, pasidalinus raktu su vienu komunikacijos dalyviu reikia susikurti naują raktą, kad galėtum komunikuoti su kitu, nes kitaip jie galės skaityti vienas kito pranešimus. Kita problema yra rakto perdavimas kitai pusei. Kadangi raktas turi būti laikomas paslaptyje, kyla problema kaip tą raktą saugiu kanalu perduoti. Visa tai lėmė, kad buvo sukurta asimetrinė šifravimo sistema, kai kiekvienam dalyviui pakanka turėti po viešą bei privatų raktą, o prieš komunikacijos užmezgimą pakanka nusiųsti savo viešą raktą, kurio net nebūtina saugoti. Visgi, po asimetrinių šifrų atsiradimo simetriniai šifrai nebuvo pamiršti dėl labai svarbios savo savybės: jie yra efektyvesni ir greitesni, negu asimetriniai šifrai. Dėl šios priežasties dauguma sistemų vienu metu naudoja tiek simetrinę, tiek ir asimetrinę kriptografiją.

Kiekvienas šifravimo algoritmas skiriasi raktų ilgiu, šifravimo bei iššifravimo sudėtingumu, greičiu, turi įvairius ribojimus šifruojamiems failams, rakto ilgiui, sunaudojamai atminčiai ir t. t.. Kiekvienas šifravimo algoritmas turi tiek pranašumų, tiek trūkumų, todėl kiekvienai situacijai kodavimo algoritmą reikia pasirinkti individualiai [8].

1.4.1. Simetriniai šifrai

1.4.1.1. DES

DES – angl. *Data encryption algorithm* – duomenų šifravimo algoritmas sukurtas Nacionalinio standartų bei technologijų instituto (NIST) 1975m. Šis algoritmas buvo vienas iš populiariausių ilgiau nei trisdešimt metų. *DES* šifravo 64- bitų dydžio blokus ir naudojo 56 bitų rakto failus [8].

Šiuo metu DES laikomas nesaugiu šifravimo metodu.

1.4.1.2. AES

AES – angl. *Advanced Encryption Standard* – duomenų šifravimo algoritmas sukurtas 2001 m. belgų kriptografų Joan Daemen ir Vincent Rijmen. Algoritmas dar kartais vadinamas pavadinimu *Rijndael*. Pagrindinis algoritmo pranašumas prieš kitus algoritmus yra platus rakto pasirinkimas. *AES* leidžia rinktis tarp 128 bitų, 192 bitų arba 256 bitų rakto, o tokio ilgio raktai lenkia daugumą kitų simetrinių šifravimo algoritmų raktų. Algoritmas šifruoja 128 bitų ilgio blokus.

Pagal NIST, AES algoritmas laikomas kaip DES algoritmo pakaitalas. Kuriant AES algoritmą didelis dėmesys buvo skiriamas į šifravimo bei iššifravimo efektyvumą atsižvelgiant tiek į programinės, tiek į aparatinės įrangos ribojimus.

Rijndael algoritmas yra vienas plačiausiai naudojamų pasaulyje, todėl kad 2000 m. buvo išrinktas AES konkurso nugalėtoju. Konkurse jis varžėsi su tokiais algoritmais kaip *Serpent, Twofish, RC6* bei *MARS.*

Algoritmas paremtas 10 raundų sistema.

Algoritmas veikia ypač sparčiai ir dėl to, kad po standartizavimo 2001 m. net ir aparatinė įranga buvo pritaikoma greitesniam AES veikimui. Nuo 2008 m. praktiškai visi *AMD* bei *Intel* gaminami procesoriai turi papildomą instrukcijų rinkinį skirtą AES operacijoms greičiau spręsti. Dėl šios priežasties AES dabartiniuose kompiuteriuose veikia ypač greit. [9]

1.4.1.3. *Blowfish*

Blowfish yra simetrinio rakto blokinio šifro algoritmas, kurį 1993 m. sukūrė Bruce Schneier ir kuris žinomas dėl savo efektyvumo ir didelio saugumo. Jis veikia 64 bitų blokais ir palaiko įvairaus ilgio raktus – nuo 32 iki 448 bitų. *Blowfish* naudoja tokią struktūrą, kai įvesties blokas padalijamas į dvi dalis, o iteracijų serija atliekama naudojant nuo rakto priklausančią raundų funkciją. Algoritmas taip pat naudoja *S-box* principu grįstą pakeitimo procesą, kuris padidina jo atsparumą žinomoms kriptografinėms atakoms.

Dėl savo patikimumo ir universalumo *Blowfish* plačiai taikomas įvairiose srityse. Jis buvo naudojamas saugiam duomenų saugojimui, apsaugant slaptą informaciją duomenų bazėse ir failų sistemose. Be to, *Blowfish* naudojamas tinklo protokoluose, kad būtų užtikrintas saugus ryšys tarp įrenginių ir sistemų. Dėl savo lankstumo, palaikančio skirtingus raktų ilgius, jis gali būti pritaikomas įvairiems saugumo reikalavimams. Nepaisant to, kad *Blowfish* buvo įdiegta daugiau nei prieš du dešimtmečius, ji tebėra patikima šifravimo schema, užtikrinanti patikimą ir veiksmingą duomenų konfidencialumo apsaugos sprendimą. [10]

1.4.1.4. *Twofish*

Twofish algoritmas buvo pasiūlytas Bruce Shneier 1998 m. Algoritmas, kaip ir kiti AES konkurso algoritmai šifruoja 128 bitų bloką, o raktas gali būti 128 bitų, 192 bitų arba 256 bitų ilgio. *Twofish* tai *Blowfish* algoritmo tęsinys.

Algoritmas veikia 16 raundų sistema. Panašiai kaip ir ankstesnis algoritmas *Blowfish, Twofish* naudoja Feistelio tinklo struktūrą kartu su tam tikra kombinacija nuo rakto priklausančių *S-box* funkcijų, nuo rakto priklausančių MDS (angl. *Maximum Distance Separable*) matricų dauginimo bei nuo rakto priklausančių bitų maišymo funkcijų [11].

1.4.1.5. *Serpent*

Serpent algoritmą pristatė Ross Anderson, Eli Biham ir Lars Knudsen. AES konkurse algoritmas liko antras (iškart po laimėtojo *Rijndael*, dabar žinomo kaip AES). Serpent šifruoja 128 bitų ilgio blokus ir gali naudoti 128 bitų, 192 bitų arba 256 bitų raktus.

Algoritmas paremtas 32 raundų, kurie šifruoja bloką, padalintą į keturias 32 bitų dalis, sistema. *Serpent* suprojektuotas taip, kad šifravimą būtų galima atlikti lygiagrečiai skirtingose gijose ir taip išnaudojamas paraleliškumas.

AES konkurse algoritmas varžėsi su konkursą nugalėjusiu *Rijndael*. *Rijndael* veikia labai panašiai, pagrindinis skirtumas yra raundų kiekis, nes *Rijndael* jų turi tik 10. Dėl to jis yra greitesnis, o kartu

ir paprasčiau atliekamas mažo dydžio informacijai šifruoti. Dėl šios priežasties konkurso laimėtojas buvo paskelbtas AES [12].

1.4.2. Asimetriniai šifrai

1.4.2.1. RSA

Algoritmas pasiūlytas 1978 metais trijų kriptografų: Rivest, Shamir ir Adleman. Tai yra bene plačiausiai paplitęs asimetrinės kriptografijos algoritmas dėl didelio efektyvumo užtikrinant saugią komunikaciją bei elektroninį parašą internete.

Algoritmas paremtas faktorizavimo uždavinio sudėtingumu. Kuo sunkesnis faktorizavimo uždavinys, tuo didesnis algoritmo sunkumas ir tuo sudėtingiau piktavaliams tokį šifrą sukompromituoti. Viešojo bei privačiojo rakto funkcijos yra du susiję pirminiai skaičiai. Šiuo metu faktorizavimo uždaviniui NIST rekomenduoja naudoti bent jau 2048 bitų arba ilgesnį modulį, kad užtikrinti adekvačią apsaugą šiai dienai ir ateityje. [13]

RSA algoritmas yra paremtas skaičių kėlimu laipsniu ribotoje skaičių skalėje (modulis iš pirminio skaičiaus). Jo apsauga slypi didelių sveikųjų skaičių faktorizavimo problemoje. Nors ir labai lengva suskaičiuoti $n = p \times q$, tačiau labai sudėtinga apskaičiuoti šį veiksmą iš kitos pusės, t. y. procesorių operacija rasti pirminius skaičiaus daugiklius pagal duotą didelį skaičių užima labai daug procesoriaus laiko ir pastangų.

1.4.2.2. *ElGamal*

ElGamal 1984 metais pristatė dvigubo rakto kriptografinį algoritmą, paremtą diskrečiojo logaritmo uždaviniu, kuris gali būti panaudotas tiek šifravimui, tiek ir elektroniniam pasirašymui. Šis viešojo ir privačiojo rakto algoritmas yra grįstas diskrečiojo logaritmo uždaviniu su dideliais pirminiais skaičiais. Algoritmas susideda iš trijų pagrindinių komponentų:

- Raktų generatoriaus;
- Šifravimo algoritmo;
- Iššifravimo algoritmo.

Vienas iš *ElGamal* trūkumų yra palyginti didelis šifro teksto dydis, palyginti su kitais algoritmais, o tai gali turėti įtakos jo veiksmingumui ir našumui tam tikrose naudojimo srityse [14].

1.4.2.3. SM2

SM2 algoritmas yra standartinis viešojo ir privataus rakto šifravimas Kinijoje. Panašiai kaip ir *ElGamal* algoritmas, jis yra paremtas elipsinių kreivių ($\mathbf{y2 = x3 + ax + b}$) uždaviniu. SM2 gali būti naudojamas užtikrinti saugų šifravimą, taip pat ir el. parašo generavimą bei tikrinimą.

Labiausiai šį algoritmą išpopuliarino Kinija, kuri pasirinko jį kaip nacionalinį standartą, taip sustiprindama jo pripažinimą ne tik Kinijoje, bet ir kitur pasaulyje.

1.5. Maišos funkcijos

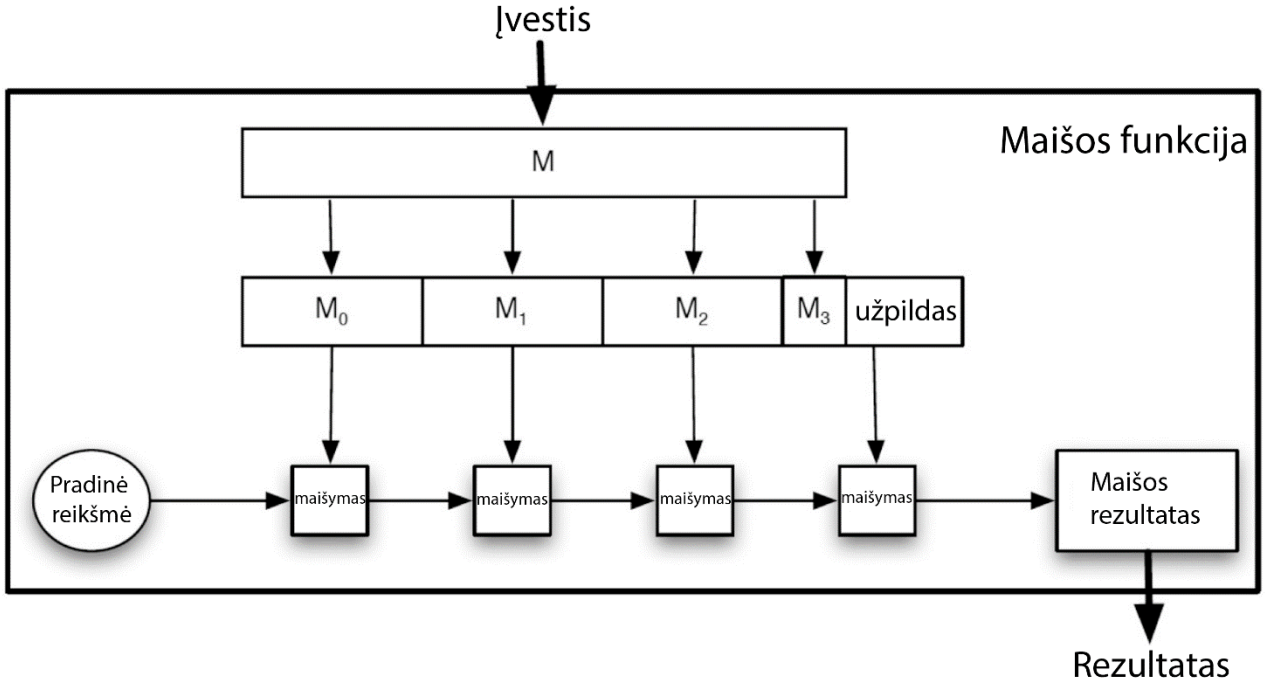
Maišos funkcija yra tokia funkcija, kuri geba duotam žinutei sugeneruoti daug trumpesnio ilgio žinutę, kuri vis dar galėtų identifikuoti pradinę žinutę. Idealiu atveju, sugeneruota funkcijos reikšmė

turėtų būti unikali ir tikti tik jos generavimo metu paduotai originaliai žinutei, tačiau praktikoje tai yra neįmanoma, nes pradinių žinučių aibė yra begalinė, o išvesčių – baigtinė.

Jei maišos funkcija skirtingoms pradinėms žinutėms grąžina tą pačią maišos reikšmę (rezultatą), įvyksta vadinamoji kolizija. Kolizija yra nemaža problema daugumoje sričių, kur vienaip ar kitaip naudojamos maišos reikšmės, todėl labai svarbu, kad maišos funkcija kaip įmanoma minimizuotų generuojamų maišos rezultatų kolizijų atvejus.

Kita labai svarbi maišos funkcijų savybė yra jų determiniškumas, kurio pavyzdys galėtų būti tai, kad jei duotoms dviems žinutėms sugeneruojamos maišos reikšmės skiriasi, tada garantuotai skyrėsi ir duotos pradinės žinutės. Dėl šios priežasties lengva patikrinti, ar dvi žinutės, arba failai, tarpusavyje skiriasi, nelyginant pilno failo reikšmės, pakanka palyginti jų maišos reikšmes. Lygiai taip pat galima lyginti ir ar failai tarpusavyje yra vienodi: jei gautas maišos rezultatas yra identiškas, tuomet galima teigti, kad failai greičiausiai yra irgi identiški.

Dauguma maišos funkcijų veikia operuodamos blokais. Algoritmo schema pažymėta **1.2 pav.** esančioje schemoje. Pradinė žinutė yra padalinama į fiksuoto ilgio blokus, o blokai vienas po kito apdorojami. Apdorojant, bloko informacija yra maišoma su vidinės algoritmo būsenos reikšmėmis, kad rezultato reikšmė turėtų didelę entropiją. Kiekviename po jo einančiame bloke yra naudojama ir praeito bloko maišos reikšmė, todėl pakeitus nors vieną bloką, kardinaliai keičiasi ir bendras rezultatas. Jei žinutėje po dalinimo lieka liekana, tuomet prie liekanos yra prijungiama užpildo reikšmė ir taip sukuriamas paskutinis normalaus dydžio blokas. [15]



1.2 pav. Maišos funkcijos veikimo schema

1.6. Egzistuojantys sprendimai saugoti duomenis paskirstytoje saugykloje

1.6.1. *CrowdStorage*

CrowdStorage Polycloud [16] yra mėnesiniu mokesčiu grįsta interneto debesų talpyklos paslauga. Pats *CrowdStorage* save lygina su tokiomis paslaugomis kaip *Amazon S3*, *Google Storage* ar *Microsoft Azure Blob storage*. Tai yra saugi privačių duomenų saugykla, kuri siūlo išskirtinai didelį pasiekiamumą ir aukštą duomenų patikimumą. Sistema yra grįsta išsidėsčiusios saugyklos pagrindu, todėl ji pati teigia, kad ji nėra pažeidžiama įprastų sutrikimų, kurie galėtų pažeisti tradicines duomenų saugyklas.

Polycloud taip pat siūlo paprastas migracijas iš kitų duomenų saugyklų tiekėjų, kad tai padaryti būtų nesudėtinga, lanksčius mokėjimus, kai mokama tik už tokią dalį duomenų, kiek tuo metu ir yra naudojama.

Pats *Polycloud* teigia, kad yra pranašesnis už įprastus duomenų saugyklos tiekėjus, nes šie naudoja didžiulius duomenų centrus, kuriems reikia dideliu patalpų, efektyvaus šaldymo, atsarginio elektros tiekimo, generatorių, kurių priežiūra išaugina tiek priežiūros, tiek ir naudojimosi kainą. Kadangi visi duomenys yra laikomi viename didžiuliame duomenų centre, neatmetama galimybė, kad dėl nenumatytos ir neapgalvotos problemos ar stichijos duomenys visgi gali būti prarasti.

Tuo tarpu *Polycloud* vietoje aparatinių saugumo mechanizmų naudoja programinius kodus. *Polycloud* išnaudoja nedidelius duomenų centrus, serverines ir kitas patalpas išsidėsčiusias įvairiose pasaulio vietose. Serveriai šiuose centruose jau iškart yra prijungti prie interneto ir yra paruošti darbui, todėl papildomas funkcionalumas saugoti duomenis jiems nesukelia didelių išlaidų. *CrowdStorage* teigia, kad toks platus serverių tinklas ir speciali programinė įranga leidžia efektyviau susidoroti su nenumatytomis problemomis, tokiomis kaip gaisrai, potvyniai ir kiti pavojai.

Kai objektas yra įkeliamas į *Polycloud* saugyklą, pirmiausia jis yra užšifruojamas su *xsalsa20poly1305* algoritmu. Taip pat, jei failas didesnis už 64-128 MB jis taip pat yra suskaldomas į mažesnius blokus. Kiekvienas blokas naudojant taisymo kodus yra suskaidomas į 40 dalių, iš kurių užtenka tik 20 dalių, kad būtų visiškai atstatytas failas. Visos šios 40 dalių yra išsiuntinėjamos į skirtingus serverius skirtingose geografinėse vietovėse. Informacija, kur randasi kiekviena iš failo dalių yra saugoma tradiciniu būdu, tačiau kadangi šie metaduomenys daug vietos neužima, juos galima nesudėtingai saugoti dvigubos ar net trigubos kopijos būdu.

Kai norima atkurti objektą, kiekviena iš 40 objekto dalių yra surandama ir pradedama siųsti atgal lygiagrečiai. Tuomet *Polycloud* sistema sulaukia dvidešimties sparčiausiai atvykusių dalių ir suformuoja pradinį failą, jį iššifruoja ir siunčia atgal galutiniam vartotojui. Kadangi sistema sulaukia tik dvidešimties iš keturiasdešimties siuntimų, ji veikia ypač greitai ir leidžia failą atkurti sparčiai, o kadangi dideli objektai yra išskaidomi į mažesnes dalis, tai didelius objektus turėtumėte gauti dar greičiau, nes jų greitis auga kartu su jų dydžiu.

Paslaugos mėnesinis mokestis – 0.004 € už kiekvieną naudojamą GB duomenų.

Tikslūs algoritmai ar skaičiai nėra atskleisti, todėl sunku pasakyti, ar sistema veikia efektyviau, negu siūlomas sprendimas, tačiau *Polycloud* funkcionalumas ne visiškai atitinka siūlomą sprendimą. *Polycloud* neturi funkcionalumo vietoje mėnesio mokesčio atsiskaityti savo paties kietojo disko vieta.

1.7. Analizės išvados

7. Objektyviai saugyklų efektyvumą galima vertinti pagal tris pagrindinius kriterijus: konfidencialumą, integralumą bei pasiekiamumą. Bet kuri saugykla privalo visus šiuos kriterijus išpildyti.
8. Kuriamos saugyklos vientisumą, o kartu ir pasiekiamumą, galima užtikrinti naudojant klaidas taisančiu fontaniniu kodu – *Raptor* kodais. *Raptor* kodai buvo pasirinkti dėl savo mažo papildomos atminties užimamo kiekio, efektyvaus šifravimo - iššifravimo prilygstančio linijiniam greičiui, algoritme lengvai pasirenkamo ar nesunkiai pakeičiamo iš vieno objekto generuojamų paketų kiekio.
9. Analizės metu rasta *RaptorQ* biblioteka realizuoja *Rapto*r kodų algoritmą, atitinka RFC 6330 aprašytą standartą ir bus naudojama saugyklos duomenų pasiekiamumo užtikrinimui.
10. Atlikus analizę buvo pastebėta, kad kuriamos saugyklos konfidencialumą geriausia užtikrinti naudojant kriptografinius algoritmus. Nuspręsta failų šifravimui naudoti simetrinį kodą *AES* su 256 bitų raktu, dėl aukšto saugumo failų dalims, saugomos trečiųjų asmenų kompiuteriuose. Raktų apsikeitimui bus naudojama asimetrinė kriptografija – ECDH viešojo ir privačiojo rakto sistema naudojant 1024 bitų raktą.
11. Jau egzistuojančių sprendimų analizės metu buvo rasta sistemų, siūlančių panašų funkcionalumą, tačiau jų paslaugas reikia pirkti už pinigus.

2. Efektyvios paskirstytos duomenų saugyklos metodas

Šiame skyriuje pateiktas efektyvios paskirstytos duomenų saugyklos metodas, pagal kurį vėliau realizuota išskirstyta duomenų saugykla.

2.1. Siūlomas metodas

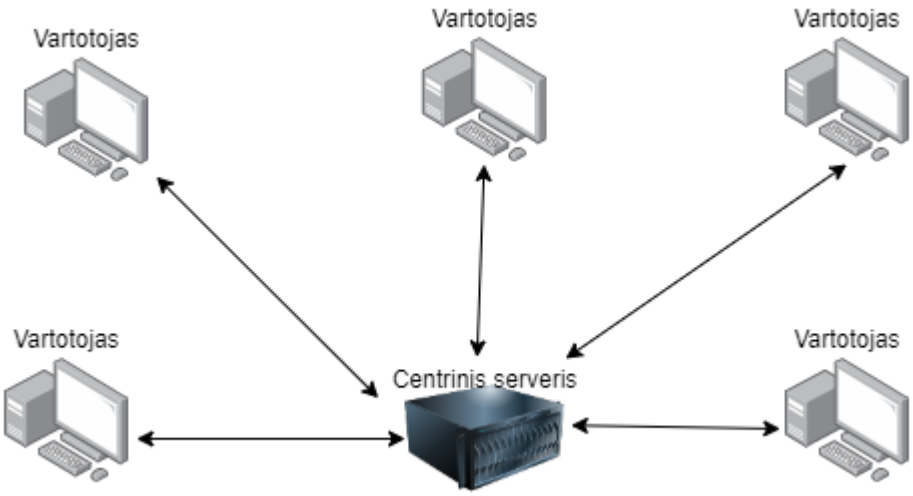
Šiame skyriuje išvardinti elementai pritaikyti efektyviam duomenų saugojimui paskirstytoje saugykloje kurti, kuri kartu buvo ir metodas – programa kompiuterio vartotojui, turinčiam prieigą tik prie savo didelės talpos kietojo disko ir norinčiam neprarasti visų savo duomenų po disko gedimo. Pasinaudojant siūlomu metodu, vartotojas galėtų įkelti savo norimą saugoti failą į sistemą, kuri savo ruoštu failą padalintų į dalis ir patalpintų į skirtingą aibę kitų vartotojų kompiuterių diskų, tokiu būdu sumažinant riziką prarasti failą po vos vieno disko gedimo.

Metodui iškelti šie reikalavimai:

- Įkeltų duomenų konfidencialumas – kiti vartotojai neturi turėti galimybės perskaityti ar pasinaudoti kito vartotojo failais;
- Įkeltų duomenų vientisumas – įkelti failai neturi pakisti ir turi išlikti tokie patys, kokie ir buvo įkelti;
- Aukštas įkeltų duomenų pasiekiamumas – metodas turi išlaikyti kuo aukštesnį duomenų prieinamumą ir galimybę juos atgauti iš saugyklos pilnus ir nepakeistus, tada kai šie duomenys yra reikalingi vartotojui. Metodas turi atsižvelgti į faktą, kad vartotojų asmeniniai kompiuteriai nėra serveriai ir paprastai nėra įjungti visą parą ir prie to prisitaikyti;
- Sistemos vartotojai už paskirstytos saugyklos paslaugas galėtų atsiskaityti mainais savo diske esančia vieta, kurioje sistema galėtų laikyti kitų sistemos vartotojų failus;
- Vartotojai turėtų būti reitinguojami įverčiu, kuris skatintų vartotojus būti vertingais sistemos mazgais ir saugoti patikėtus duomenis.

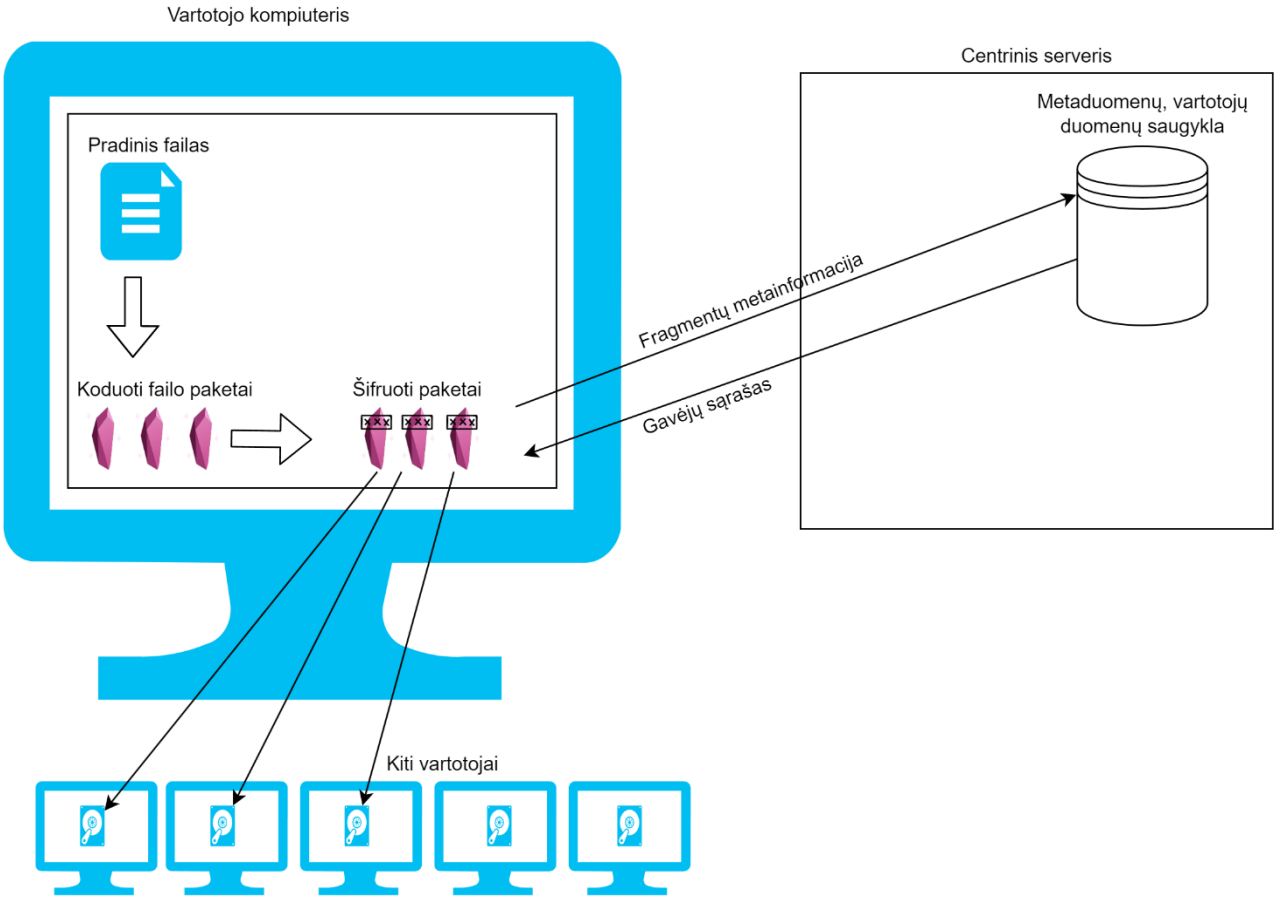
2.2. Efektyvios paskirstytos duomenų saugyklos koncepcinė vizija

Duomenų saugyklos sistema turėtų leisti įprastiems kompiuterio vartotojams gauti disko talpos failams laikyti su dideliu patikimumu, mainais už tai saugant kitų vartotojų duomenis (**2.1 pav.**). Pradedant naudotis programa vartotojas turėtų nurodyti, kokio kiekio talpos bei kokio aukštumo patikimumo vartotojas reikalauja ir pagal koeficientus sistema apskaičiuotų, prie kokio kiekio savo asmeninio disko talpos vartotojas turėtų suteikti prieigą sistemai.



2.1 pav. Efektyvios paskirstytos duomenų saugyklos koncepcinė vizija

Tarkime, kad vartotojas nori įkelti naują failą į saugyklą (žr. **2.2 pav.**). Mazgo programa vartotojo kompiuteryje pasirinktą failą užkoduoja fontaniniu kodu į tam tikrą kiekį fragmentų (failų). Po to mazgo programa šiuos failus užšifruoja simetriniu šifru naudojant vartotojo pasirinktą slaptą raktą. Gauti failai arba paketai yra siunčiami kitiems vartotojams, kad šie saugotų jį savo diske. Kadangi failas šifruotas ir be to, kiekvienas vartotojas turi tik mažą failo dalį, vartotojai svetimo failo fragmento iššifruoti ir peržiūrėti negali jokiu būdu. Centrinė sistema savo duomenų bazėje saugo informaciją apie tai, kokiam vartotojui išsiuntė kurio failo fragmentą, kad vėliau būtų galima atsekti, kurie vartotojai kuriuos fragmentus savo diske turi.

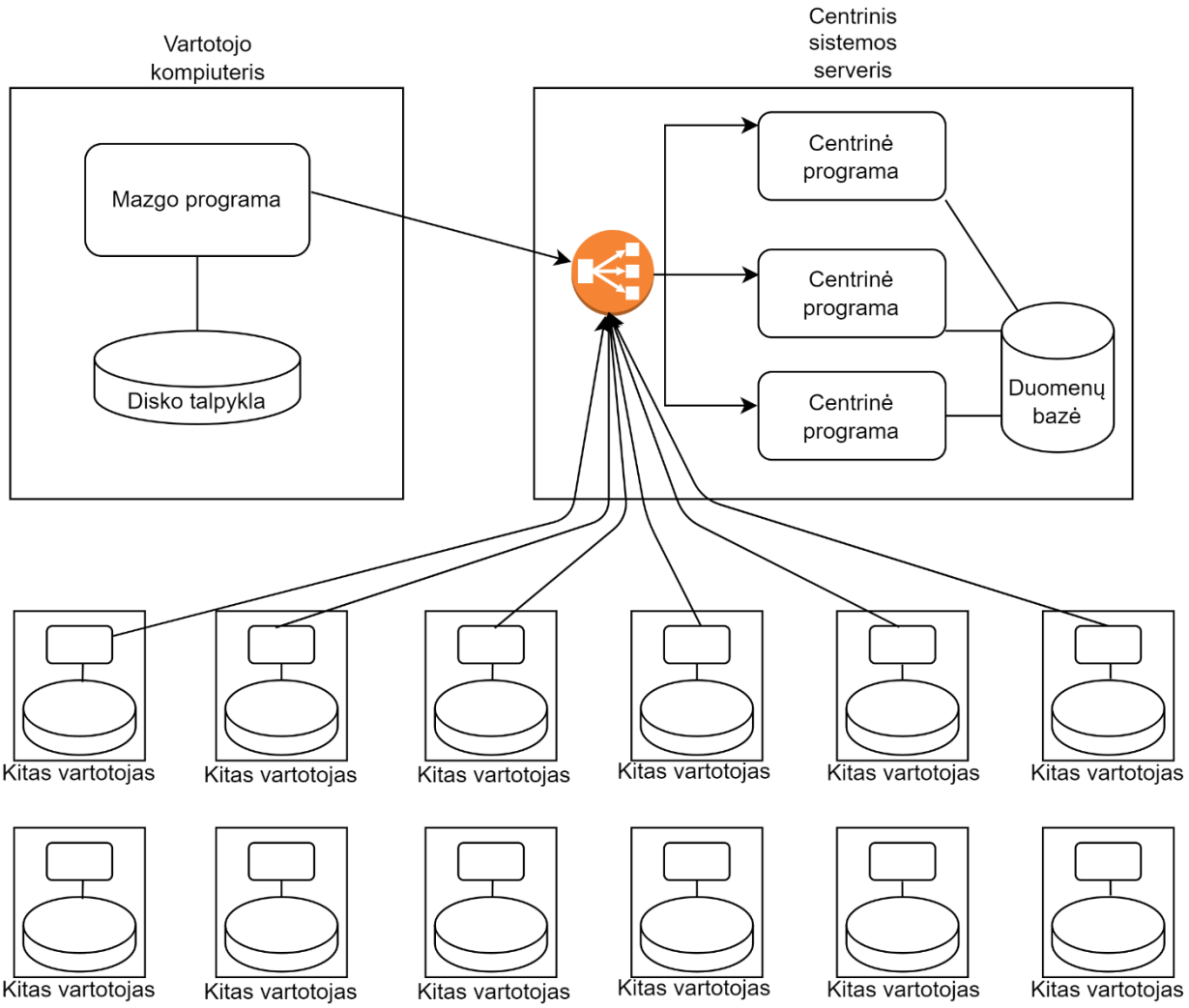


2.2 pav. Bylos įkėlimo į paskirstytą saugyklą koncepcinė schema

Kai vartotojas vėliau nori šį failą parsisiųsti, pirmiausia jis turi pranešti apie tai sistemai. Sistema, gavusi prašymą apie norimą parsisiųsti failą, savo duomenų bazėje peržiūri, kurie vartotojai turi nurodyto failo fragmentus. Tuomet sistema pareikalauja iš kiekvieno vartotojo, kad šie atsiųstų savo turimą failo fragmentą. Vartotojo kompiuteryje esanti mazgo programa gavusi fragmentus pradeda tikrinti, ar įmanoma iš fragmentų atkurti originalų failą. Kiekvienas fragmentas yra iššifruojamas, tikrinama jo maišos reikšmė ir jei viskas sutampa, paketas yra naudojamas originaliam failui atstatyti. Jei failas atkūrimas sėkmingas, sistema baigia savo darbą. Jei ne, rodomas klaidos pranešimas ir vartotojas gali bandyti failą atstatyti kitu metu, kai bus prisijungę daugiau failo dalis savo kompiuteriuose turinčių vartotojų, arba naudotis avariniu atstatymu, kai failo fragmentai yra kaupiami vartotojo kompiuteryje.

2.3. Sistemos architektūra

Sistema susideda iš dviejų tipų komponentų: centrinės programos, kuri atsakinga už failų valdymą, bei mazgo programos, kuri diegiama vartotojų kompiuteriuose. Šie du komponentai tarpusavyje nuolatos turi komunikuoti (žr. **2.3 pav.**).



2.3 pav. Sistemos architektūros diagrama

2.3.1. Centrinė programa

Centrinė programa yra atsakinga už visos sistemos veikimą. Ji nuolatos komunikuoja su nutolusiais mazgais – vartotojų programomis. Ji taip pat priima užklausas iš mazgo programų ir siunčia duomenis vartotojams apie jų įkeltus failus, priima informaciją apie naujai bandomą įkelti failą ar organizuoja seniau vartotojo įkelto failo parsisiuntimą.

Centrinės programos serveris taip pat turi duomenų bazę, kurioje saugoma saugykloje saugomų duomenų metainformacija, tokie duomenys kaip vartotojai, jų slaptažodžiai, vartotojų įkelti failai, įkeltų failų fragmentų lokacijos mazguose, taip pat fragmentų failų pavadinimai.

Centrinės programos serveris yra kritiškai svarbus, todėl būtina turėti atsarginių serverių, kurie galėtų perimti srauto apdorojimą, jei vienas iš serverių nustotų funkcionuoti. Tokia architektūra kartu leidžia ir paskirstyti srautą tarp centrinių serverių, jei vienu metu sistema naudojasi didelis kiekis vartotojų.

2.3.2. Mazgo programa

Mazgo programą privalo parsisiųsti ir įsidiegti kiekvienas vartotojas, kuris nori naudotis sistema. Programa privalo turėti prieigą prie vartotojo failų sistemos, dar vadinamos disko talpykla. Mazgo programa nuolatos komunikuoja su centriniu serveriu ir parsisiunčia kitų vartotojų norimus saugoti

failus bei, centrinei programai informavus, siunčia kitam vartotojui jam reikalingus failų fragmentus. Mazgo programa taip pat leidžia vartotojui suteikia prieigą naudotis saugykla. Ši programa leidžia vartotojams prisijungti, autentifikuotis, peržiūrėti įkeltus failus, įkelti naujus failus ar paprašyti parsisiųsti anksčiau įkeltą failą.
Įkeliant naują failą, mazgo programa koduoja pasirinktą failą į paketus, tuomet šiuos paketus šifruoja ir siunčia kitiems vartotojams į mazgo programas, kad jos išsaugotų šias šifruotų failų dalis vartotojų diskuose. Parsisiunčiant failą iš saugyklos, mazgo programa priima šifruotus paketus iš kitų vartotojų, juos iššifruoja ir bando atstatyti originalų failą.

2.4. Duomenų bazė

Centrinė programa saugo įkeltų failų metainformaciją, kurią sudaro vartotojų duomenys (**2.1 lentelė**), failų duomenys (**2.2 lentelė**) ir failo fragmentų duomenys (**2.3 lentelė**).

Projektuojamos duomenų bazės struktūra pateikta **2.4** pav.

2.1 lentelė. Vartotojų duomenys

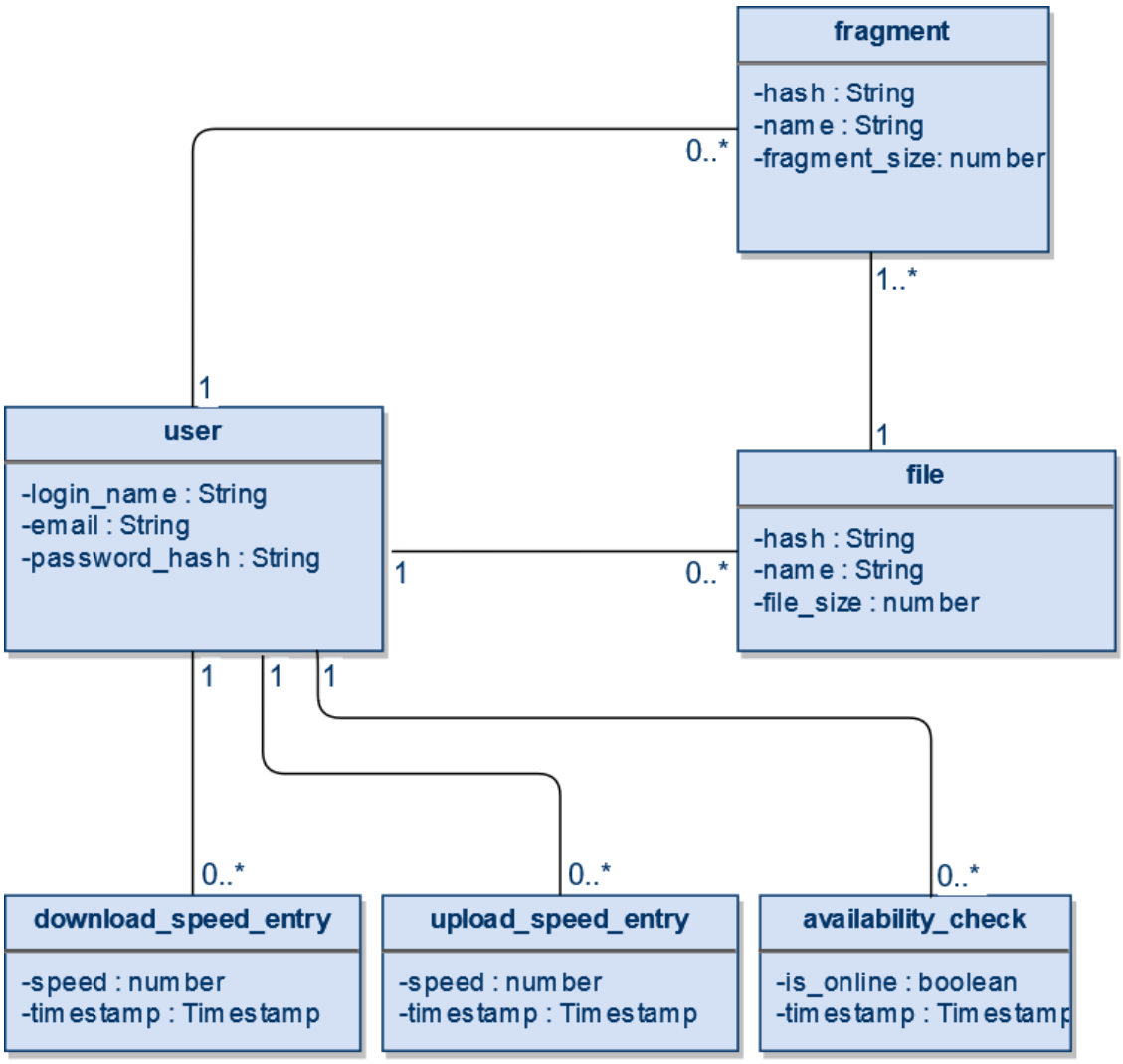
Saugomi duomenys	**Paskirtis**
Prisijungimo vardas	Prisijungimo vardas leidžia identifikuoti vartotoją
Elektroninis paštas	Elektroninis paštas reikalingas tokiu atveju, jei vartotojas pamirštų savo slaptažodį
Slaptažodžio reikšmė	Slaptažodis reikalingas autorizuoti kiekvieną sistemos vartotoją
Duomenys, reikalingi apskaičiuoti naudingumo kategoriją	Tokie duomenys kaip interneto išsiuntimo greitis, parsiuntimo greitis bei mazgo prieinamumas, skirti nustatyti mazgo naudingumą

2.2 lentelė. Failų duomenys

Saugomi duomenys	**Paskirtis**
Pavadinimas	Failo pavadinimas skirtas failo atkūrimui
Vartotojas, kuriam failas priklauso	Skirtas nustatyti, kam priklauso konkretus failas
Maišos reikšmė	Maišos reikšmė leidžia patikrinti failo vientisumą

2.3 lentelė. Failo fragmentų duomenys

Saugomi duomenys	**Paskirtis**
Failas, kuriam fragmentas priklauso	Skirtas nustatyti, kuriam failui priklauso konkretus fragmentas
Vartotojas, kurio diske fragmentas išsaugotas	Leidžia surasti, kur ieškoti konkretaus fragmento
Maišos reikšmė	Maišos reikšmė leidžia patikrinti fragmento vientisumą



2.4 pav. Projektuojama duomenų bazės struktūra

2.5. Vartotojų naudingumo įvertis

Sistemos vartotojų kompiuteriai, skirtingai nei serveriai, nėra laikomi įjungti visą parą. Vartotojų kompiuteriai būna įjungti tik tada, kai vartotojai jais naudojasi. Dėl to kenčia jų kompiuterių duomenų saugyklos pasiekiamumas. Kad užtikrinti stabilesnį failų pasiekiamumą, turi būti imamasi veiksmų, kad būtų galima įvertinti kiekvieno vartotojo kompiuterio arba mazgo patikimumą.

Šiai paskirčiai centrinis serveris kas nustatytą laiko intervalą bando kreiptis į kiekvieną iš mazgų ir rezultatus saugo duomenų bazėje. Pagal šiuos kreipimosi rezultatus sistema gali nustatyti, kokią laiko dalį, vidutiniškai, šis mazgas yra pasiekiamas (**2.4 lentelė**). Pagal šią metriką galima vertinti mazgus bei pačius vartotojus į kategorijas, kurios nurodo mazgo naudingumo laipsnį.

Mazgo naudingumo laipsniui turi įtakos ir mazgo duomenų išsiuntimo greitis, todėl išsiuntimo greitis (**2.5 lentelė**) matuojamas kiekvieną kartą siunčiantis iš mazgo failą.

Bendras naudingumo įvertis susideda iš 80% dalies patikimumo įverčio ir 20% siuntimosi greičio įverčio. Pagal naudingumo įvertį vartotojai skirstomi į keturias grupes:

– Labai patikimi;

– Patikimi;
– Dalinai patikimi;
– Nepatikimi.

Balas apskaičiuojamas pagal šią formulę:

$$B = P \cdot 0{,}8 + D \cdot 0{,}15 + U \cdot 0{,}05; \quad (1)$$

čia P – patikimumo įvertis, gaunamas pagal **2.4 lentelė**;
D – parsisiuntimo greitis, gaunamas pagal **2.5 lentelė**;
U – įkėlimo greitis, gaunamas pagal **2.5 lentelė**.

2.4 lentelė. Patikimumo įverčio apskaičiavimo lentelė

Įvertis	**Teigiamų patikrinimų dalis (%)**
10	90 - 100
9	80 - 89,99
8	70 - 79,99
7	60 - 69,99
6	50 - 59,99
5	40 - 49,99
4	30 - 39,99
3	20 - 29,9
2	10 - 19,99
1	0 - 9,99

2.5 lentelė. Tinklo greičio įverčio apskaičiavimo lentelė

Įvertis	**Tinklo greitis (Mbps)**	**Aprašymas**
10	500+	Ypatingai geras interneto greitis, leidžia dideliu greičiu siųstis didelio dydžio failus
9	201-500	
8	101-200	
7	76-100	Geras interneto greitis, leidžia greitai siųstis įvairius failus bei kitus resursus.
6	51-75	
5	31-50	
4	21-30	Patenkinamas interneto greitis, leidžia siųstis didesnius failus ir greitai naršyti internete
3	11-20	Mažesnis nei vidutinis greitis, tačiau naršymui internete pakanka
2	6-10	Lėtas greitis, failai siunčiasi lėtai, gali lėčiau veikti ir paprasti veiksmai internete
1	0-5	Ypatingas lėtas greitis, sudėtinga atlikti net paprastus uždavinius

Pagal vartotojo naudingumo įvertį priklauso tam vartotojui leidžiama į saugyklą įkelti failų dydžio suma. Tarkime, jei esate patikimas vartotojas, tuomet galite įkelti daugiau duomenų, negu tie

vartotojai, kurie savo kompiuterį įjungtą laiko trumpiau. Tokiu būdu kiekvienas mazgas skatinamas būti kuo patikimesniu. Taip pat, vartotojo įkeliama informacija stengiamasi įkelti į kuo panašesnio įverčio vartotojų saugyklas, kokio įverčio yra ir pats vartotojas. Tarkime, jei vartotojas yra dalinai patikimas, tai ir jo įkelto failo fragmentai bus keliami pagal prioritetą į tų vartotojų diskus, kurie priklauso dalinai patikimų vartotojų įverčio grupei.

2.6. Avarinis failo parsisiuntimas iš neaktyvių mazgų

Jei failo fragmentai įkeliami į nepatikimus mazgus, arba tiesiog retai esančius aktyviais mazgus, ir failo siuntimosi momentu nėra pakankamo kiekio prisijungusių vartotojų, kurie tiektų failo fragmentus, failą galima parsisiųsti avariniu režimu. Avarinis režimas leidžia vartotojui pačiam kaupti norimo failo fragmentus iš kitų vartotojų, o surinkus reikiamą jų kiekį, atstatyti failą.

Tarkime, vartotojo failas gali būti padalintas į fragmentus, o fragmentai išsaugoti 100 skirtingų mazgų, kurių aktyvumas stipriai skiriasi dėl skirtingų laiko zonų, todėl tam tikru laiko momentu vartotojui pavyksta paprašyti failo kai prisijungę tik 60 vartotojų. Kadangi 60 fragmentų nepakanka pilno failo atstatymui, failas nebus parsiųstas ir sistema siūlys palaukti ir bandyti iš naujo vėliau. Jei tai tęsiasi per ilgai, vartotojas gali rinktis paprašyti gauti visus prieinamus failo fragmentus. Tuomet vartotojas į savo diską parsisiunčia 60 failo fragmentų, palaukia keletą valandų, parsisiunčia dar 20 fragmentų ir t. t. Rezultate, vartotojas turi 80-90 failo fragmentų, kurių pilnai pakanka pradinio failo atstatymui, nors vienu metu maksimaliai prisijungdavo tik 60 mazgų.

2.7. Reguliari failo fragmentų patikra mazguose ir piktavaliai vartotojai

Kiekvienas sistemos vartotojas tampa atsakingas ne tik už savo, bet ir už kitų vartotojų failus. Dėl šios priežasties yra reikalingas kiekvieno vartotojo atsakingumas, saugant savo diską ne tik nuo fizinių, bet ir nuo kitokių pavojų. Papildomai, pateikiamas ir sistemos metodas patikrinti, ar fragmentai vis dar saugomi svetimų vartotojų diskuose sėkmingai.

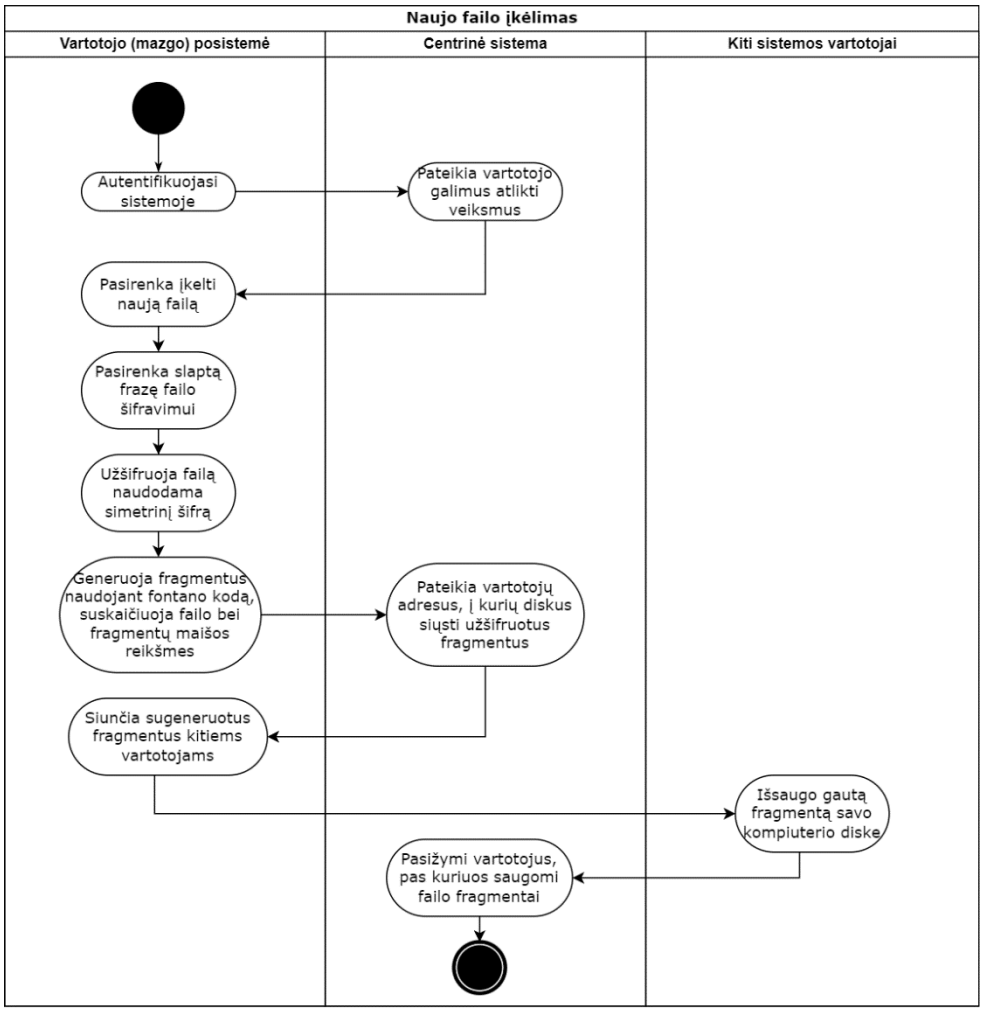
Šis metodas susideda iš reguliaraus (kas mėnesį) kiekvieno iš failų fragmentų patikrinimo diske. Patikrinimas inicijuojamas iš centrinio serverio, kuris pastebėjęs, kad suėjo tam tikras terminas ir kad mazgo vartotojas aktyvus, išsiųstų užklausą patikrinti diske saugomų fragmentų maišos reikšmes. Papildomai, jei šios maišos reikšmės sutampa, tam tikri failai atsitiktiniu būdu būtų persiunčiami į centrinį serverį, kad centrinis serveris patikrintų jų maišos reikšmes. Tokiu būdu apsisaugoma nuo piktaviliškų vartotojų, kurie imituoja teisingą atsakymą, net nepatikrinę failų savo diske.

Jei pastebima, kad vartotojo diske reikalingų failų nėra arba failai yra nepasiekiami, pakeisti ar kitaip sugadinti, vartotojas yra įvertinamas neigiamai. Žemo įverčio vartotojams gali būti apribota galimybė naudotis saugykla t. y. suteikiama paskutinė galimybė parsisiųsti įkeltus failus, o po tam tikro laiko nebeleidžiama naudotis. Taip pat, vartotojas, kurio fragmentai buvo sugadinti, yra įspėjamas, kad bendras pasiekiamų fragmentų kiekis sumažėjo. Tokiu atveju vartotojas gali pasirinkti, kaip nori elgtis. Jis gali arba pasitikėti, kad dar liko pakankamas kiekis fragmentų ir nesiimti jokių veiksmų, arba atlikti failo pakartotinį įkėlimą, galbūt su didesniu pasikliautinumu.

2.8. Failų kodavimas

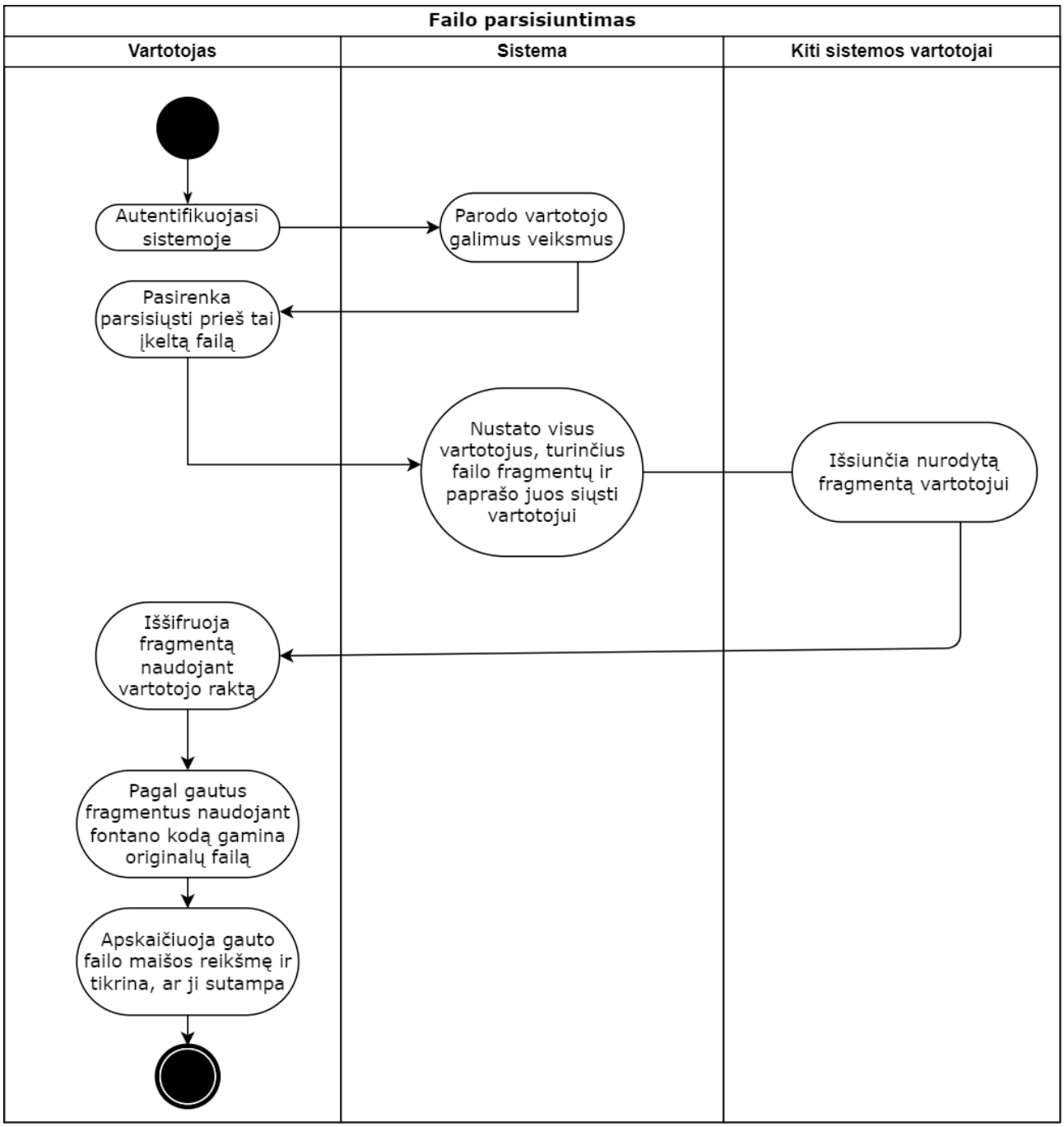
Naujo failo įkėlimas prasideda nuo vartotojo programos, kurioje vartotojas pirmiausia turi autentifikuotis (žr, **2.5 pav**.). Po autentifikacijos ir prisijungimo sistema pateikia vartotojui galimus atlikti veiksmus. Vartotojas pasirenka įkelti failą. Mazgo programa, esanti vartotojo kompiuteryje,

pagamina fragmentus (paketus) pagal pradinį failą. Po to kiekvieną iš jų užšifruoja vartotojo pasirinktu raktu. Centrinis serveris atsiunčia sąrašą vartotojų, kurie yra pasiruošę priimti failo dalis. Sukurti fragmentai yra išsiuntinėjami šiems vartotojams į jų kompiuteriuose veikiančias mazgo programas. Mazgo programos išsaugo gautą fragmentą vartotojų talpyklose, o centrinė sistema pasižymi duomenų bazėje, kurie vartotojai turi failo fragmentus, kad vėliau galėtų šią informaciją pateikti.



2.5 pav. Naujo failo įkėlimo scenarijaus veiklos diagrama

Failo parsisiuntimas prasideda nuo to, kad vartotojas turi prisijungti per savo kompiuteryje esančią mazgo programą bei autentifikuotis (žr. **2.6 pav.**). Po prisijungimo sistema atsiunčia galimus atlikti veiksmus. Vartotojas pasirenka parsisiųsti saugykloje saugomą failą. Tada pagal vidinę duomenų bazė sistema nustato visus vartotojus, kurie turi failo fragmentų ir nurodo juos siųsti pirminiam vartotojui. Vartotojų kompiuteriuose esanti mazgo programa reaguoja į prašymą ir išsiunčia vartotojo talpykloje esantį šifruotą fragmentą. Vartotojas, gavęs fragmentus, iššifruoja juos savo raktu ir bando juos sujungti į pradinį failą, naudojant fontaninio kodo algoritmą. Tai daroma tol, kol pavyksta atkurti pradinį failą.



2.6 pav. Failo parsisiuntimo scenarijaus veiklos diagrama

2.9. Vartotojai, kurie nusprendžia nebesinaudoti paskirstyta saugykla

Vartotojai, kurie nusprendžia nebesinaudoti saugyklos teikiamomis paslaugomis privalo apie tai pranešti ir suteikti galimybė visiems vartotojams, kurie saugo bent po vieną fragmentą jų diske, parsisiųsti savo failus. Jei vartotojai nepasinaudoja tokia galimybe per mėnesį (pvz. nusprendžia, kad saugykloje dar pakanka fragmentų pas kitus vartotojus), leidžiama ištrinti jų fragmentus ir nustoti naudotis saugykla.

Taip pat, įvykus disko gedimui vartotojas turėtų nedelsiant informuoti centrinį serverį, kuris informuotų paveiktus vartotojus, kad jie galėtų nuspręsti, kaip elgtis su laikomais failais. Lygiai tas

pats galioja ir norint perkelti turimus fragmentus į kitą kompiuterį ar kitą vietą, reikia informuoti centrinį serverį, kad jis atliktų reikalingus veiksmus ir vartotojas negautų neigiamo įverčio.

2.10. Efektyvaus parsisiuntimo algoritmas

Kadangi sistemoje saugomam failui atkurti nereikalingi visi failo fragmentai ir pakanka tik dalies iš jų, kad būtų galima atkurti pradinį failą, tai, kad pagreitinti failų gavimą, naudojamas specialus algoritmas. Pradžioje paprašoma visų failo fragmentų iš visų vartotojų, kurie turi nors vieną fragmentą. Tuomet pagal gautus fragmentus mazgo programoje esantis algoritmas bando atstatyti pradinį failą. Po kiekvieno naujai gauto fragmento testuojama, ar pavyko atkurti failą. Jei ne, tuomet fragmentai ir toliau siunčiami ir jungiami į pradinį failą, tol, kol pavyksta jį atkurti. Iškart po sėkmingo failo atstatymo, likę siuntimai yra nutraukiami, todėl failas atstatomas greičiau, nei laukiant visų fragmentų siuntimo pabaigos.

2.11. Sprendimo paskirtis ir tinkamumas

Dėl šifravimo ir duomenų konfidencialumo užtikrinimo paskirstyta duomenų saugykla tinkama naudojimuisi tiek ir tokiomis situacijomis, kai sistemos vartotojai vienas kito nepažįsta, tiek ir kai sistema naudojama vidinėje aplinkoje, tarkime tarp įmonės ar mokyklos kompiuterių.

Jei sistema naudojama vidinėje aplinkoje ir joje nėra saugomi kritinio konfidencialumo reikalaujantys duomenys, vienas iš variantų būtų atsisakyti failų šifravimo, kad būtų galima pasiekti sąlyginai aukštesnį sistemos našumą. Kadangi atskiras vartotojas neturi prieigos prie didelio kiekio tam tikro failo fragmentų, pilnai atstatyti pradinį failą iš duomenų, esančių viename kompiuteryje galimybės nėra. Visgi, tokiu atveju keli piktavaliai vartotojai gali sujungti savo turimus fragmentus į didesnį rinkinį ir tokiu būdu bandyti atkurti pradinį failą. Kai kurie fontaniniai kodai pasižymi gebėjimu dalinai atkurti tam tikras failo dalis, net ir neturint kritinio kiekio fragmentų. Papildomai dar įvertinus ir sąlyginai nedidelį šifravimo užimamą laiką, galima prieiti išvados, kad nenaudoti šifravimo paskirstytoje saugykloje yra visiškai nerekomenduojama praktika.

Imlias laikui procedūras (šifravimą bei kodavimą) sistema atlieka vartotojų kompiuteriuose, dėl to centriniam sistemos serveriui nėra reikalinga galinga aparatinė įranga. Be to, šifravimo raktas nėra saugomas centriniame serveryje, jį turėtų prisiminti pats vartotojas, todėl dar mažiau sumažėja galimybė piktybiškai atkurti failus.

Visos failo atliekamos procedūros (šifravimas, kodavimas, maišos reikšmių skaičiavimas) yra atliekamos patikimais ir efektyviais algoritmais, kurie puikiai veikia daugumoje įprastų kompiuterių. Vartotojams nėra reikalinga jokia papildoma aparatinė įranga, nes didelė dalis naudojamų algoritmų yra linijinio sudėtingumo, kai kurie algoritmai yra projektuoti atsižvelgiant į ribotus techninius parametrus ar net turi papildomą instrukcijų rinkinį procesoriuje operacijai sparčiau atlikti.

Kitas svarbus aspektas yra centrinis serveris. Nors ir jame yra saugoma informacija apie kiekvieno iš failų fragmentų lokaciją tinkle, failai yra šifruoti, o raktas centriniame serveryje nėra saugomas, dėl to net ir sukompromitavus centrinį serverį, nei vieno originalaus saugyklos failo pasiekti nepavyks. Dėl šios priežasties centrinis serveris mažiau domina užpuolikus, dėl to ir mažėja rizika saugyklai tapti nepasiekiama. Visgi, jei dėl tam tikrų priežasčių centrinis serveris taptų nepasiekiamas, saugykla kurį laiką neveiktų, todėl būtina pasirūpinti centrinio serverio saugumu.

Veiksmai, reikalingi užtikrinti tinkamą centrinio serverio veikimą:

- Prižiūrėti centrinio serverio darbą, tikrinti laisvą disko vietą, procesoriaus bei atminties užimtumą, reaguoti į įvykusias klaidas ar anomalijas;
- Turėti atsarginį centrinį serverį, į kurį būtų galima peradresuoti užklausas, jei pirminis serveris neveiktų;
- Reguliariai daryti centriniame serveryje saugomas metaduomenų kopijas;
- Sukurti dokumentuotas schemas ir procesus, kaip turėtų elgtis darbuotojai, ištikus tam tikram incidentui.

2.12. Efektyvaus duomenų saugojimo paskirstytoje saugykloje metodo išvados

1. Pasiūlytas metodas efektyviai saugoti bei parsisiųsti duomenis iš paskirstytos duomenų saugyklos.
2. Sistemos mazgai, vartotojų kompiuteriai, nėra serveriai, kurie veikia ištisą parą, todėl į tai buvo atsižvelgta ir vartotojai yra skirstomi į patikimumo klases, kurios juos reitinguoja pagal naudingumo lygį.
3. Efektyvaus saugojimo paskirstytoje saugykloje metodas leidžia parsisiųsti kritinius failus avariniu, lėtuoju būdu, kai failui atstatyti reikalingos dalys yra kaupiamos vartotojo kompiuteryje.
4. Peržvelgti dažniausiai pasitaikantys sistemos piktavalių scenarijai ir pateiktas planas, kaip sistema elgiasi tokiose situacijose.

3. Efektyvaus saugojimo paskirstytoje saugykloje metodo prototipo realizacija

Esminių metodo veikimo vietų demonstracijai yra sukurtas metodo prototipas. Prototipui realizuoti naudotas *Docker* (versija 20.10.11) aplinkos konteineris. Jo pagalba galima imituoti autonomiškos sistemos veikimą, tačiau jie nereikalauja atskiro hipervizoriaus, todėl sistema mažiau apkraunama.

Prototipas programuotas *Python* (versija 3.8) programine kalba. Failų šifravimui bei iššifravimui reikšmių skaičiavimui buvo naudojamas *OpenSSL* programinis įrankis (versija 3.0.0). Failo kodavimui *Raptor* fontaniniu kodu buvo naudojama *Rust* programinės kalbos biblioteka *RaptorQ*.

Failų bei jų dalių metainformacijai saugoti buvo naudojama reliacinė duomenų bazė *MariaDB* (versija 10.7.1). Iš esmės šioje vietoje buvo galima rinktis ir tarp kitų reliacinių duomenų bazių, tačiau ši buvo pasirinkta dėl sąlyginai didesnio našumo, mažo kompiuterio resursų naudojimo bei to, kad yra nemokama ir viena populiariausių pasaulyje.

3.1. Sistemos diegimas

Minimaliam sistemos veikimui, sistema privalo veikti bent trijuose skirtinguose įrenginiuose arba konteineriuose. Vienas iš jų yra centrinės programos serveris, o likusieji – klientų kompiuteriai (žr. **3.1 pav**.).

Kliento kompiuteriuose esanti mazgo programa susideda iš skirtingų servisų.

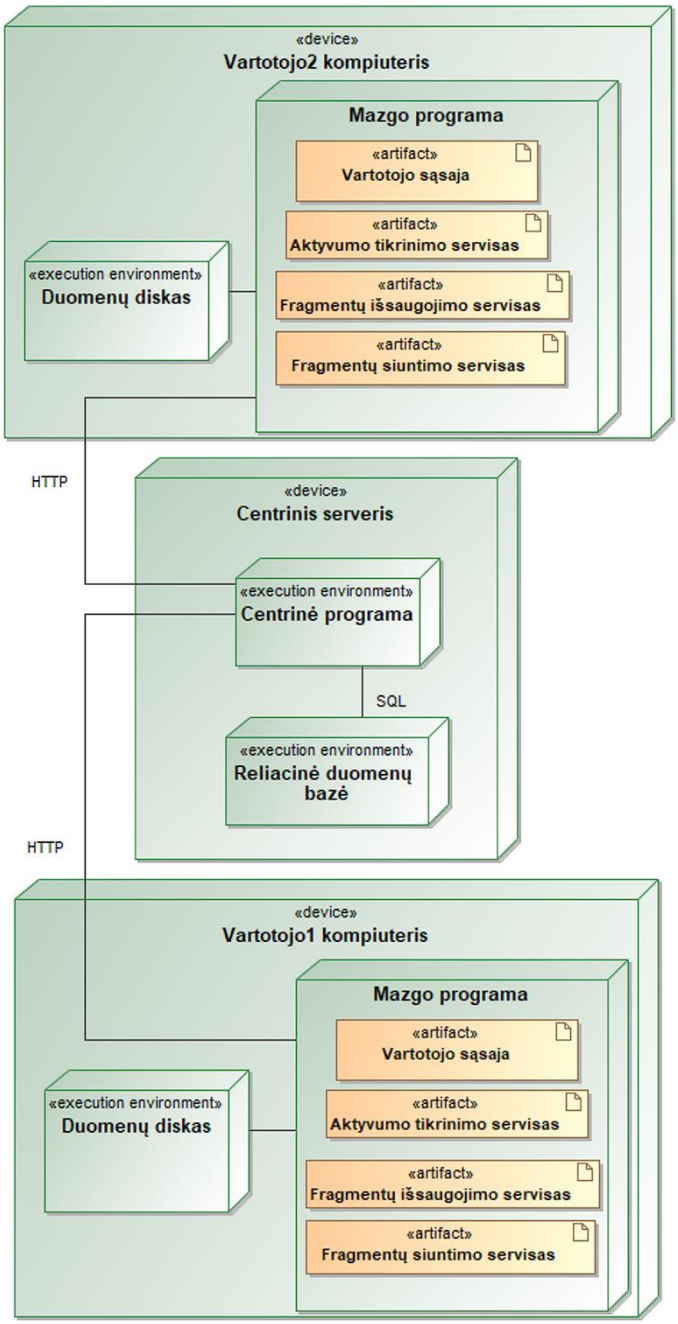
Vartotojo sąsajos posistemė suteikia prieigą vartotojui naudotis sistema: įkelti ir parsisiųsti įkeltus failus.

Aktyvumo tikrinimo servisas atsako į aktyvumo užklausas ateinančias iš serverio. Pagal tai vėliau serveris skaičiuoja naudingumo balus ir atitinkamai gali leisti padidinti taisymo (papildomų) blokų kiekį įkeliamiems failams.

Fragmentų išsaugojimo servisas koordinuoja fragmentų išsaugojimą kitų vartotojų kompiuteriuose. Jis skirsto fragmentus ar jų grupes konkretiems sistemos vartotojams, prieš tai patikrinęs jų aktyvumą ir galimybės išsaugoti tam tikrą failą.

Fragmentų siuntimo servisas atsako į užklausas ateinančias iš mazgo programos esančios vartotojų kompiuteriuose. Šios užklausos inicijuojamos tada, kai vartotojas paprašo iš saugyklos parsisiųsti anksčiau įkeltą failą. Gavęs tokią užklausą, centrinis serveris tikrina, kurie vartotojai turi nurodyto failo fragmentų. Sudaręs tokių vartotojų sąrašą, centrinis serveris tikrina kiekvieno iš jų pasiekiamumą. Išfiltravęs sąrašą ir turėdamas tik šiuo metu prisijungusius vartotojus, fragmentų siuntimo serveris inicijuoja fragmentų siuntimą iš vartotojų diskų tiesiai į failo savininko kompiuterį.

Verta pabrėžti, kad vartotojo mazgo programa vienu metu gali atlikti abu vaidmenis, tiek ir leisti įkelti failus į saugyklą, tiek ir leisti parsisiųsti duomenis iš jos, todėl vartotojų kompiuteriuose esančios programos yra vienodos, nepaisant nuo naudojimo paskirties.

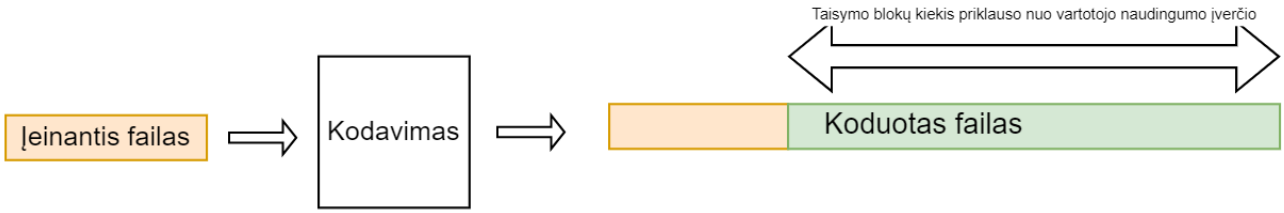


3.1 pav. Paskirstytos saugyklos diegimo diagrama

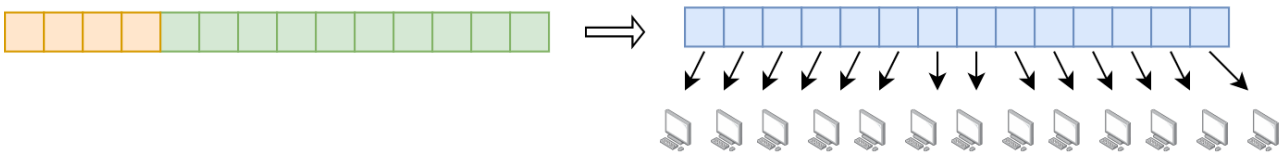
3.2. Kodavimas

Kodavimo posistemėje failas yra apdorojamas naudojantis Rust programavimo kalbos biblioteka *RaptorQ*. Pradžioje yra pasirenkami pradiniai parametrai, tokie kaip bloko dydis, taisymo blokų kiekis, kuris priklauso nuo vartotojo naudingumo įverčio. Taip pat apskaičiuojama pradinio failo maišos funkcijos reikšmė, bei nustatomas pradinio failo dydis. Dalis šios informacijos siunčiama ir išsaugoma į reliacinę duomenų bazę esančią centrinėje sistemoje. Po to pradedamas kodavimas (žr. **3.2 pav.**). Gaunami koduoti fragmentai, kurių bendras dydis visada yra didesnis už pradinio failo dydį. Dalių kiekis priklauso nuo to, koks kiekis vartotojų tuo metu turi galimybę priimti failą. Jei failo dalių kiekis yra didesnis negu vartotojų kiekis, kurie gali priimti failą, dalis fragmentų yra grupėmis į bendrą failą. Tai kartojama tol, kol dalių kiekis atitinka vartotojų kiekį arba yra mažesnis. Po to

kiekviena iš šių dalių yra užšifruojama simetriniu šifru. Tik tuomet dalys yra išsiuntinėjamos vartotojams gavėjams, kurių sąrašą pateikia centrinis serveris (žr. **3.3 pav.**).



3.2 pav. Failo kodavimas



3.3 pav. Dalių išsiuntinėjimas vartotojams

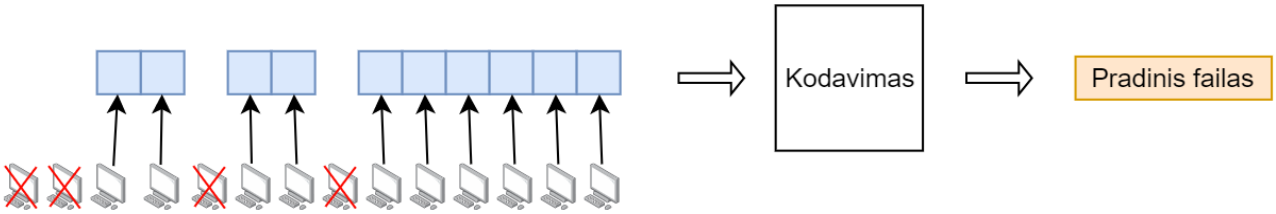
3.3. Šifravimas

Nors, iš esmės, didėjant sistemos vartotojų kiekiui, vienoje vietoje surinkti kritinį kiekį fragmentų ir atstatyti pradinį failą nesankcionuotai tikimybė bei rizika mažėja, papildomai prieš siunčiant failo fragmentus kitiems vartotojams į diskus, šie fragmentai yra užšifruojami simetriniu šifru, o tik po to išsiunčiami. Simetrinio šifro raktas yra parenkamas dar vartotojo kompiuteryje, todėl sistema jo nežino ir nesaugo, tik operuoja jau šifruotais failais. Šifravimui yra pasirinkta biblioteka *OpenSSL*, šifravimas atliekamas AES algoritmu naudojant 256 bitų ilgio raktą su CBC režimu.

3.4. Failo atstatymas

Sistemos prototipas leidžia vartotojui paprašyti sistemos atkurti ir atsiųsti prieš tai įkeltą failą. Šis scenarijus prasideda iš vartotojo kompiuteryje esančios vartotojo sąsajos komponento. Failo užklausa perduodama į centrinį sistemos serverį ir jame ši užklausa pradedama apdoroti. Pradžioje duomenų bazėje randama ieškomo failo metainformacija ir vartotojų sąrašas, kurie turi to failo fragmentų. Tuomet tikrinamas kiekvieno vartotojo prieinamumas ir galimybė parsisiųsti koduotą bei šifruotą fragmentą. Fragmento iššifravimas atliekamas vartotojo kompiuteryje su vartotojo raktu, todėl centriniam serveris simetrinių raktų nesaugo. Po iššifravimo fazės prasideda pradinio failo atstatymo fazė. Natūralu, kad ne visų vartotojų asmeniniai kompiuteriai tuo metu pasiekiami, todėl ne visi failo fragmentai bus atsiunčiami (žr. **3.4 pav.**). Kadangi failas buvo koduotas fontaniniu kodu, tai nėra problema ir gali būti prarastas sąlyginai didelis kiekis fragmentų, tačiau failas vis dar gali būti atstatomas sėkmingai. Tai priklauso nuo pertekliškumo laipsnio pasirinkto generuojant koduotus paketus. Jei gautas pakankamas kiekis fragmentų, pradinis failas atstatomas sėkmingai.

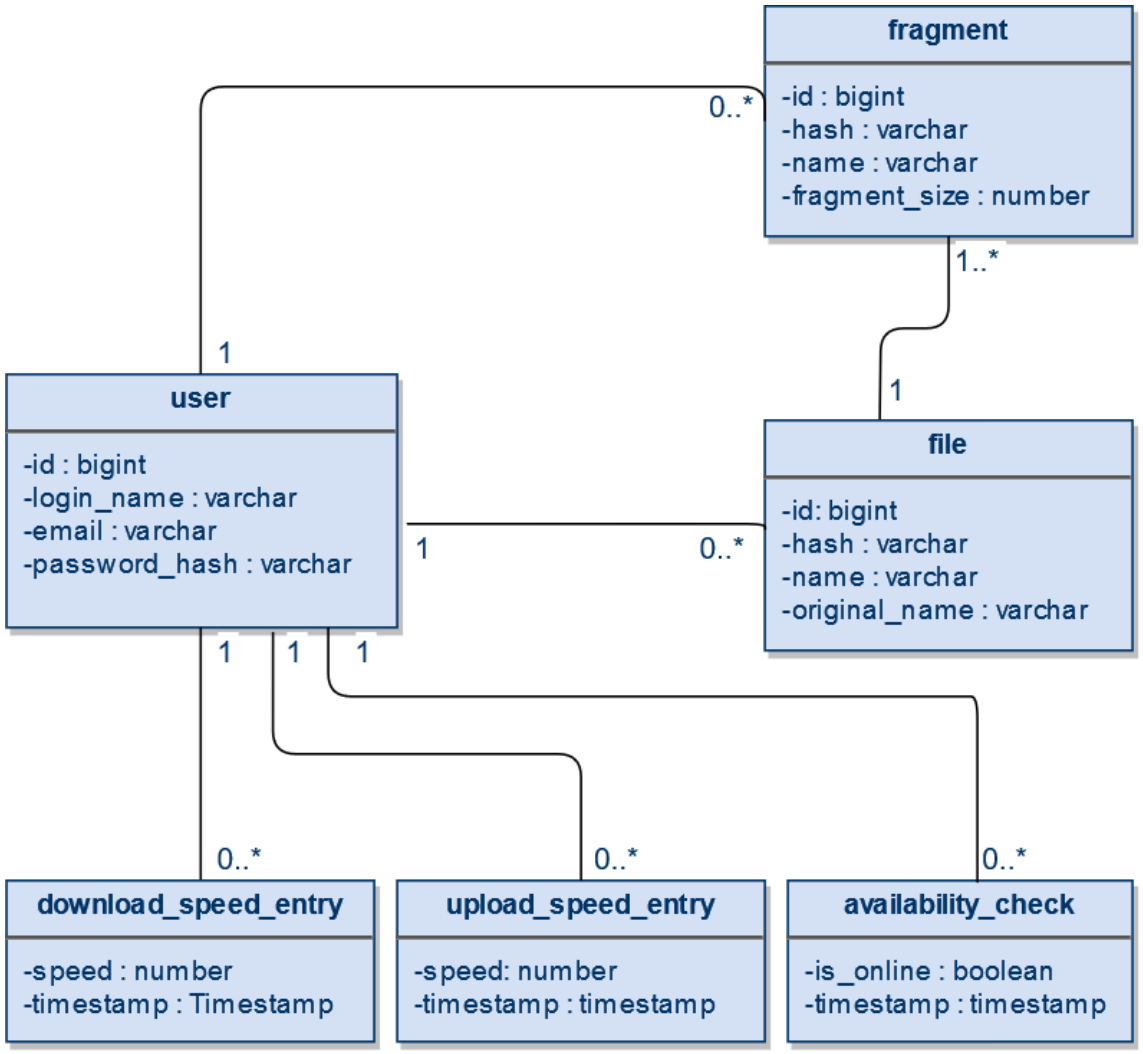
Jei failo visgi nepavyko atstatyti, vartotojas gauna pranešimą, kad tuo metu nėra pakankamai prisijungusių failo fragmentus laikančių vartotojų. Vartotojas gali po tam tikro laiko bandyti operaciją atlikti iš naujo, arba iniciuoti avarinį atstatymą, kurio metu kauptų fragmentus savo kompiuteryje.



3.4 pav. Failo gavimo iš parskirstytos saugyklos schema

3.5. Centrinės programos duomenų bazė

1. *MariaDB* reliacinėje duomenų bazėje viso yra šešios duomenų lentelės (žr. .



3.5 pav. Centrinės programos duomenų bazės schema:

Informacija apie vartotojus – *user*:

1. Vartotojo identifikatorius;
2. Vartotojo paskyros vardas;
3. Vartotojo el. paštas;
4. Vartotojo slaptažodžio maišos reikšmė.

Informacija apie saugomus failus – *file*:

1. Failo identifikatorius;
2. Failo pavadinimas;
3. Failo maišos reikšmė;
4. Failo originalus pavadinimas;
5. Kodavimo bloko ilgis;
6. Kodavimo blokų kiekis;
7. Pradinis failo dydis;
8. Vartotojas, kuriam priklauso failas.

Informacija apie saugomų failų fragmentus – *fragment*:

1. Fragmento identifikatorius;
2. Failo fragmento pavadinimas;
3. Failo fragmento dydis;
4. Failo fragmento maišos reikšmė;
5. Failas, kuriam priklauso fragmentas;
6. Vartotojas, kurio diske išsaugotas failas.

Informacija, skirta vartotojo naudingumo balui nustatyti (parsisiuntimo greitis) – *download_speed_entry*:

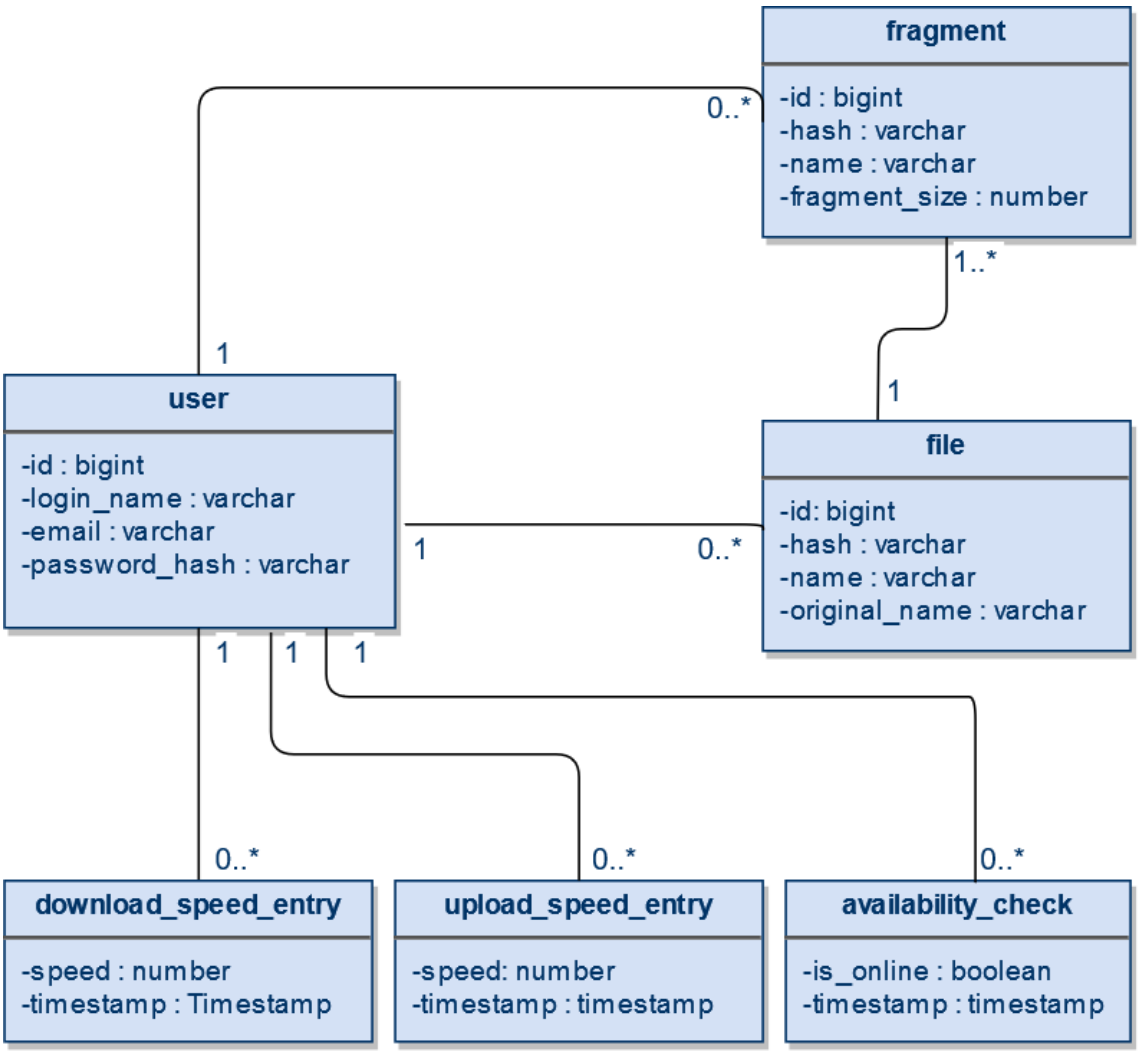
1. Siuntimo greitis;
2. Laiko žyma;
3. Vartotojas, kuriam priklauso metrika.

Informacija, skirta vartotojo naudingumo balui nustatyti (išsiuntimo greitis) – *upload_speed_entry*:

1. Įkėlimo greitis;
2. Laiko žyma;
3. Vartotojas, kuriam priklauso metrika.

Informacija, skirta vartotojo naudingumo balui nustatyti (prieinamumas) – *availability_check*:

1. Laiko žyma;
2. Ar vartotojas buvo pasiekiamas nurodytu metu;
3. Vartotojas, kuriam priklauso metrika.



3.5 pav. Efektyvaus saugojimo paskirstytoje saugykloje metaduomenų duomenų bazės struktūra

3.6. Realizacijos išvados

1. Praktiškai realizuotas metodo dalyje suprojektuotos sistemos prototipas, kuris geba tiek įkelti, tiek atstatyti failus iš paskirstytos saugyklos.
2. Failo įkėlimas realizuotas koduojant fontaniniais kodais *Raptor*, tada šifruojant simetriniu šifru pagal vartotojo pasirinktą raktą ir išsiunčiant kitiems sistemos vartotojams į jų kietuosius diskus.
3. Vientisumas užtikrinamas naudojantis vienkryptėmis maišos funkcijomis. Maišos funkcijų rezultatas skaičiuojamas tiek pradiniam failui, tiek kiekvienai sukoduotai ir užšifruotai daliai.
4. Metainformacija apie įkeltus duomenis bei sistemos vartotojus yra laikoma *MariaDb* reliacinėje duomenų bazėje centriniame serveryje.

4. Tyrimas

Šiame skyriuje aprašytas pasiūlyto metodo testavimas ir tyrimo rezultatai bei išvados. Atliekant testavimą į paskirstytą saugyklą buvo saugomi failai ir matuojamos įvairios metrikos. Pagrindiniai naudojami įrankiai – *Linux* operacinė sistema, *Python* programavimo kalba.

Atliktas tyrimas turi parodyti failų saugykloje esančių failų vientisumą, pačios sistemos našumą ir jo priklausomybę nuo skirtingų failo ypatybių.

4.1. Našumo testavimas

Našumo testavimas atliekamas asmeniniame kompiuteryje, kurio specifikacijos pateiktos žemiau. Testavimo metu bus lyginamos skirtingų įkėlimo bei failo atstatymu operacijų užimamos laiko trukmės. Atskirai bus matuojamas laikas vykdant kodavimo, šifravimo, maišos skaičiavimo bei metaduomenų įrašymo operacijas.

Darbo stoties, kurioje atliktas testavimas, techniniai parametrai:

- Windows 10 Pro 64-bitų operacinė sistema
- AMD Ryzen 5 2600 6 branduolių, 12 gijų procesorius
- 16GB DDR4 2400Mhz operatyvinė atmintis
- 465GB SSD diskas

Testavimo metu į sistemą buvo įkelti 277,06 KB, 5,25 MB, 39,95 MB, 616,22 MB ir 1,56 GB dydžio failai. Buvo matuotas laikas, per kurį failas koduojamas į dalis naudojantis fontaniniu kodu, laikas, per kurį failas užšifruojamas simetriniu šifru, taip pat laikas, per kurį apskaičiuojamos failo maišos reikšmės bei laikas, per kurį metainformacija išsaugoma į duomenų bazę.

Taip pat lentelėje pateiktas duomenų perkėlimo greitis, tačiau verta pabrėžti, kad siuntimo tinklu operacijos greitis pateiktas ne konkrečios sistemos, o pagal statistinį Lietuvoje esančių įmonių turimos prieigos prie interneto greitį. Be to, nebuvo vertintas persiuntimo metu generuojamos antraštės perkėlimo greitis, nes augant siuntinio dydžiui, antraštė sudaro vis mažesnę paketo dalį ir siunčiant didelius failus antraštės dydis didelės įtakos neturi.

Pagal Lietuvos Statistikos 2020 m. atliktą tyrimą [1], didžioji dalis įmonių Lietuvoje (30,6%) turi prieigą prie interneto, kurio greitis tarp 100 ir 500 Mbit/s. Šis rodiklis yra dar aukštesnis, jei imtį pakeistume tik į įmones, kurios turi bent 250 darbuotojų. Tokiu atveju net 40,4% įmonių turi prieigą prie interneto tinklo, kurio greitis tarp 100 ir 500 Mbit/s. Dėl šios priežasties tyrimui atlikti naudota interneto greičio vidutinė šio įverčio reikšmė. Taigi, statistinis vidutinis interneto greitis Lietuvos įmonėse 2020 m. = $\frac{100+500}{2} = 300$ Mbit/s.

Kita vertus, jei sistema sudiegta vietiniame tinkle, tikėtina, kad tinklo greitis tarp mazgų gali būti dar greitesnis negu interneto greitis.

Kiekvienas iš testų atliktas 10 kartų ir lentelėje (**4.1 lentelė**) pateikta vidutinė vertė.

4.1 lentelė. Failų įkėlimo ir atstatymų iš paskirstytos saugyklos užimamo laiko tyrimo rezultatai

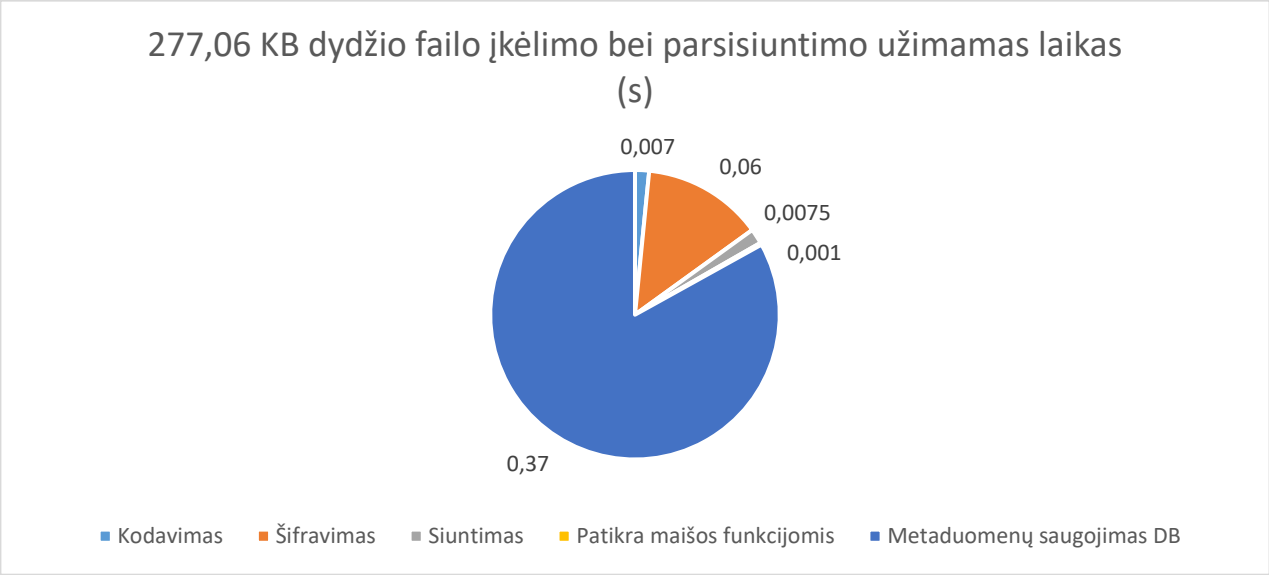
Failas bei operacija	**Paketų kiekis**	**Failų kiekis**	**Kodavimas į dalis (s)**	**Šifravimas (s)**	**Fragmentų perkėlimas tinklu (s)**	**Patikra maišos funkcijomis (s)**	**Metaduomenų duomenų bazėje apdorojimas (s)**	**Bendras apdorojimo laikas (s)**	**Bendras laikas (s)**
277,06 KB įkėlimas	204	102	0,004	0,03	0,0007	0,0006	0,31	0,05	0,05
277,06 KB atstatymas	203	102	0,003	0,03	0,0007	0,0005	0,06	0,04	0,04
5,25 MB įkėlimas	3934	562	0,02	0,14	0,14	0,01	0,72	0,78	0,92
5,25 MB atstatymas	3934	562	0,01	0,13	0,14	0,01	0,17	0,26	0,4
39,95 MB įkėlimas	1677	559	0,15	0,27	1,06	0,06	0,68	1,53	2,59
39,95 MB atstatymas	1677	559	0,15	0,26	1,06	0,06	0,14	0,63	1,69
616,22 MB įkėlimas	12925	517	2,79	0,81	16,43	0,51	1,32	20,14	36,57
616,22 MB atstatymas	12925	517	2,39	0,37	16,43	0,51	0,19	12,37	28,8
1,56 GB įkėlimas	33453	507	28,56	1,68	41,6	1,04	2,42	45,78	87,38
1.56 GB atstatymas	33453	507	17,79	0,76	41,6	1,05	0,18	35,96	77,56

Pagal lentelės duomenis, žemiau pateikiamos užimamo laiko diagramos (**4.1 pav. - 4.8 pav.**). Pagal diagramas matome, kad į saugykla įkeliant ar pasisiunčiant failus iki 5,25MB dydžio, didžiausią laiko tarpą užima duomenų apie įkeliamus fragmentus saugojimas duomenų bazėje. Pažvelgus į lentelę, taip pat matosi, kad metaduomenų saugojimas labiausiai priklauso nuo fragmentų failo kiekio, ir tik mažiau priklauso nuo failo dydžio, todėl didėjant įkeliamo failo dydžiui, ši operacija procentaliai užima vis mažiau laiko.

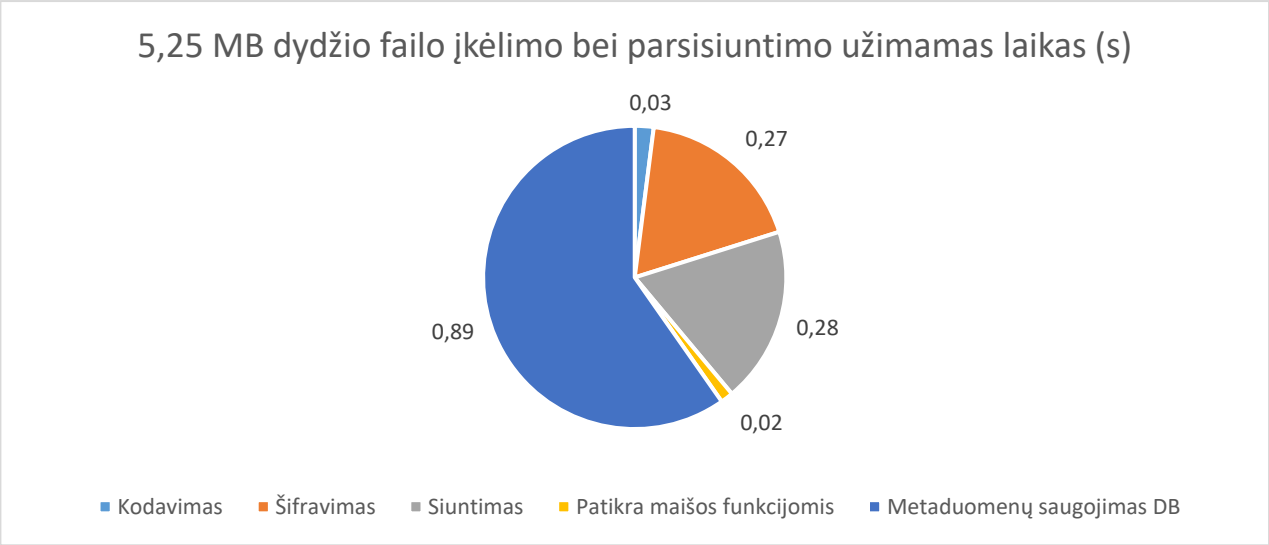
Didesniems failams ilgiausią laiko tarpą užima duomenų perkėlimas tinklu (1,56 GB bei 616,22 MB failams – daugiau nei pusę bendro laiko, o 39,95 MB failui – apie pusę laiko). Dar viena ryški

tendencija yra šifravimo operacijos trukmė mažiems bei itin mažiems failams. Itin mažiems failams ši operacija yra antroji pagal ilgumą (iškart po metaduomenų saugojimo DB), o 5,25MB failui – trečioji (ją pralenkia tik metaduomenų saugojimas DB bei perkėlimas tinklu).

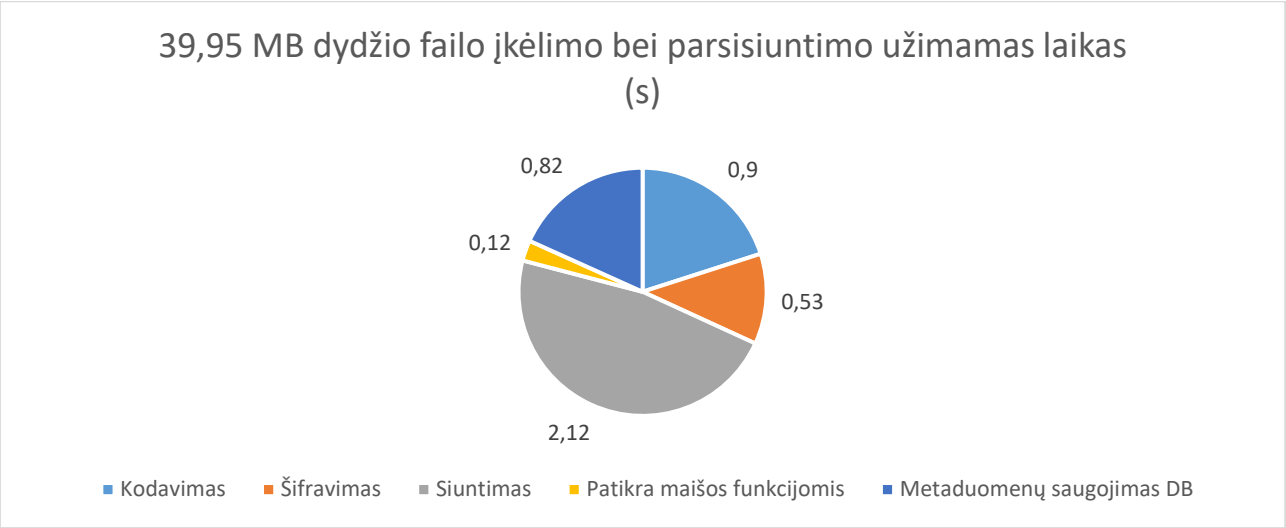
Dideliems failams (virš 616,26 MB dydžio) reikšmingą dalį laiko užima kodavimo operacija (sąraše pagal laiko trukmę ji eina iškart po duomenų perkėlimo tinklu). Lentelėje matome, kad dideliems failams ypač kyla paketų kiekis fragmente, o nuo to auga ir laikas, reikalingas koduoti failą.



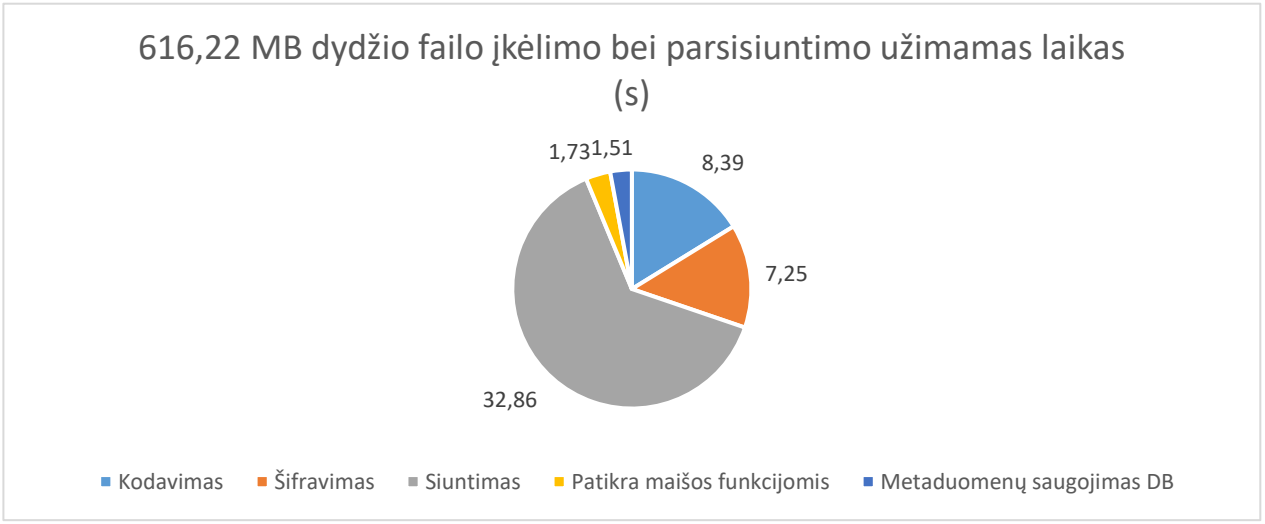
4.1 pav. 277,06 KB dydžio failo įkėlimo bei parsisiuntimo užimamas laikas (s)



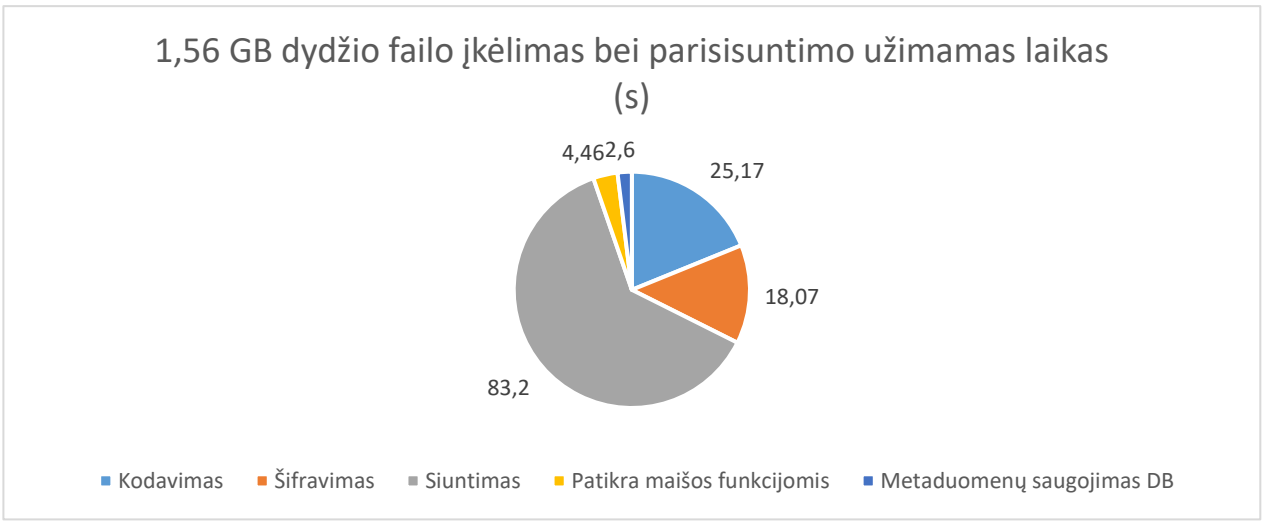
4.2 pav. 5,25 MB dydžio failo įkėlimo bei parsisiuntimo užimamas laikas (s)



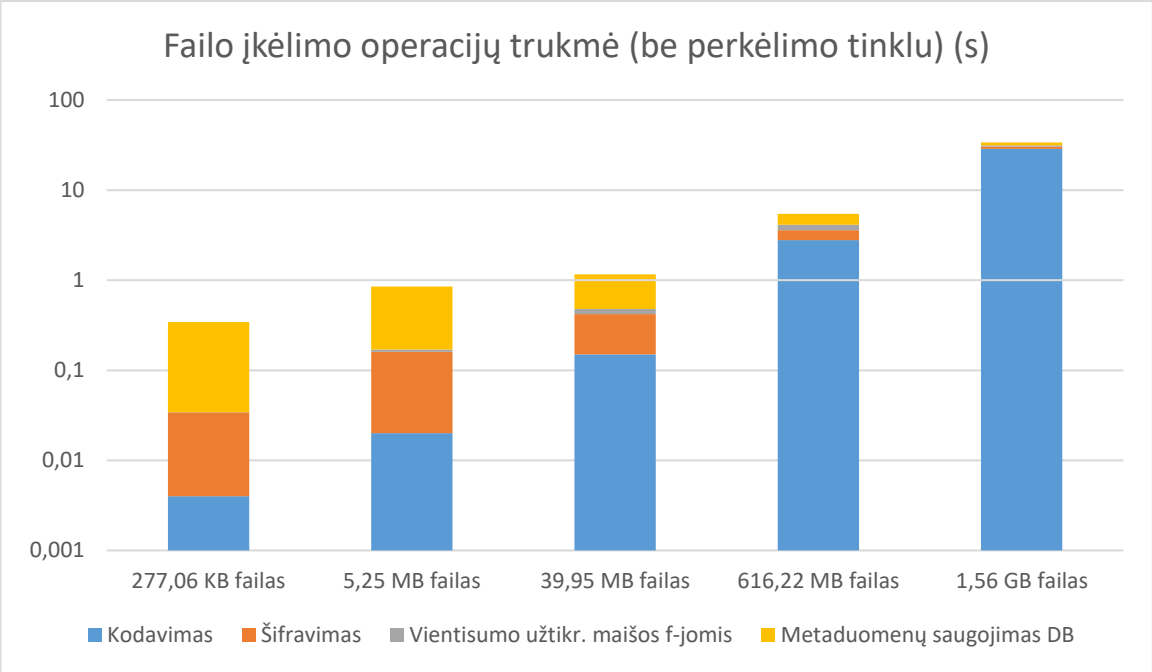
4.3 pav. 4.439,95 MB dydžio failo įkėlimo bei atstatymo užimamas laikas (s)



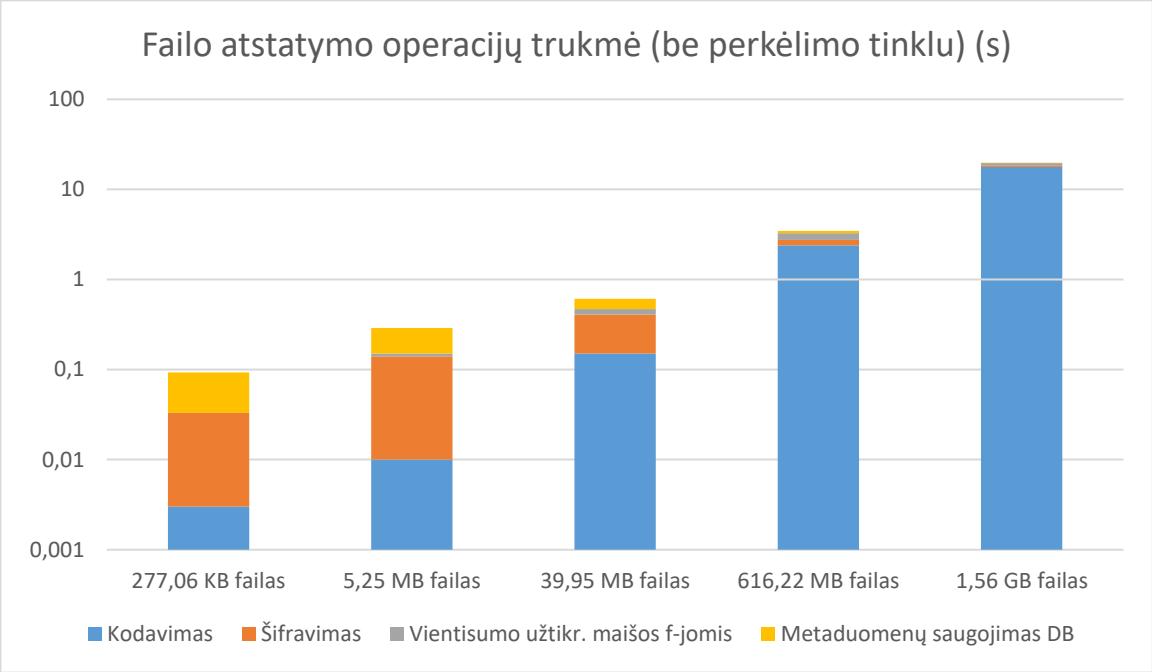
4.5 pav. 5,25 MB dydžio failo įkėlimo bei parsisiuntimo užimamas laikas (s)



4.6 pav. 1,56 GB dydžio failo įkėlimo bei parsisiuntimo užimamas laikas (s)

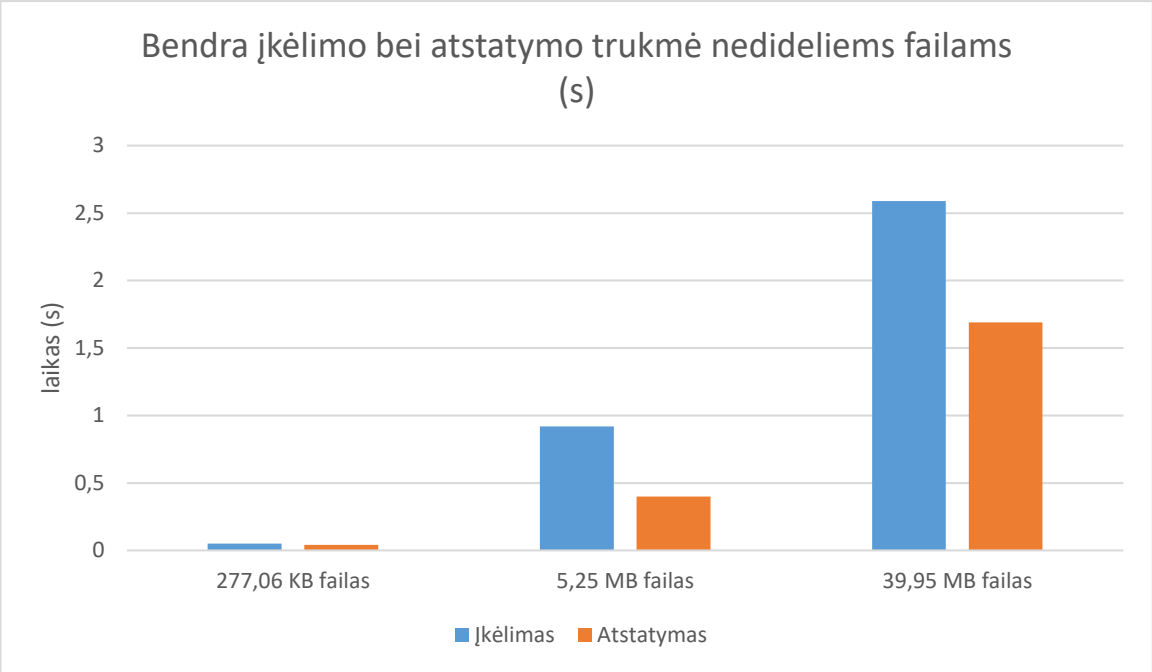


4.7 pav. Failo įkėlimo skirtingų operacijų trukmė, neskaičiuojant perkėlimo tinklu (s)

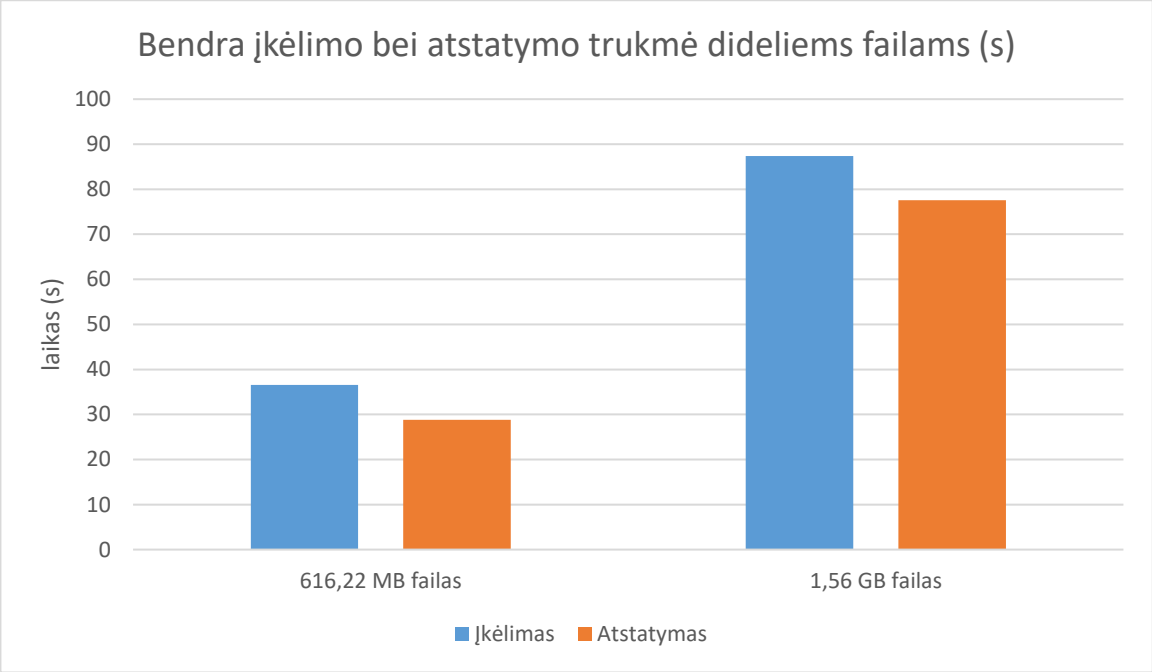


4.8 pav. Failo atstatymo skirtingų operacijų trukmė, neskaičiuojant perkėlimo tinklu (s)

Lentelėje palyginę bendras įkėlimo bei failo atstatymo užimamas trukmes, galima daryti išvadą, kad failo įkėlimas į saugyklą užima daugiau laiko nei failo atstatymas iš saugyklos. Mažiems bei itin mažiems failams įkėlimo trukmė trunka beveik dvigubai ilgiau, negu atstatymo. Didžiausią įtaką tam daro metaduomenų saugojimo operacija, kuri užtrunka ženkliai ilgiau failo saugojimo metu, negu atstatymo. Didesniems failams metaduomenų saugojimas tokios didelės laiko dalies nesudaro ir bendros įkėlimo bei atstatymo trukmės pasidaro panašios, tačiau atstatymas vis dar kartais užima net 20% mažiau laiko, negu įkėlimas.



4.9 pav. Nedidelio dydžio failų bendras įkėlimo bei atstatymo laikas (s)



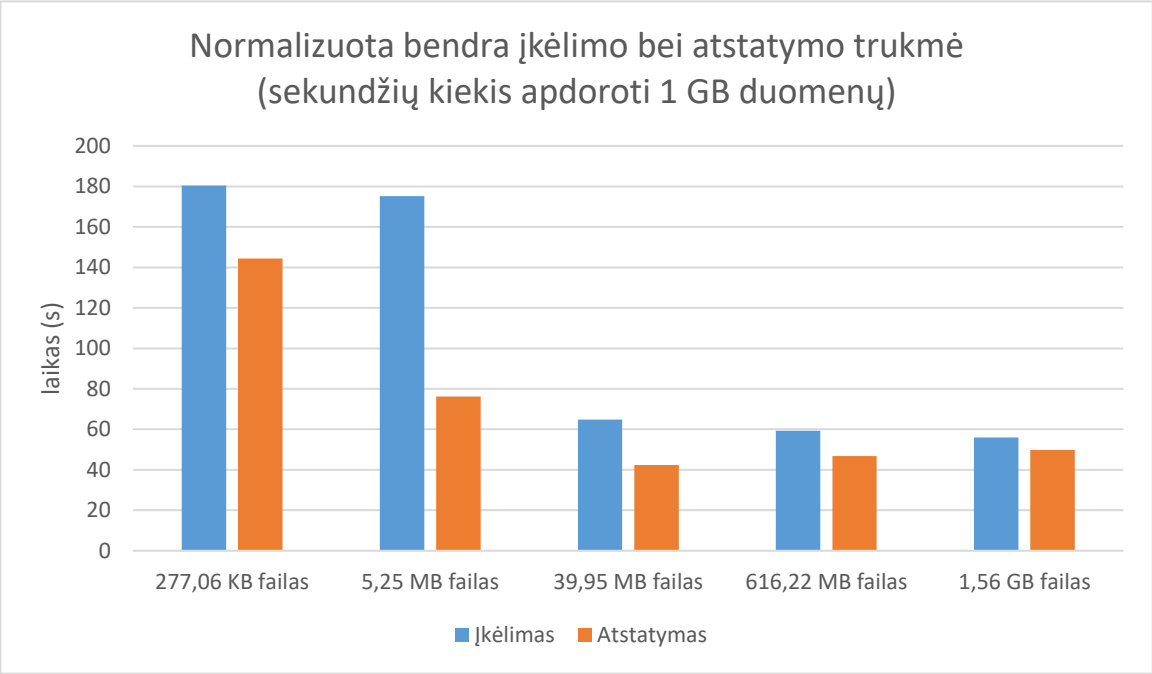
4.10 pav. Didelio dydžio failų bendras įkėlimo bei atstatymo laikas (s)

Kad ištirtume saugyklos efektyvumą, taip pat turime atlikti normalizuotų failo įkėlimo bei atstatymo trukmių analizę. Įkelti didesnį failą dažniausia užtruks ilgiau, negu mažesnį, tačiau vertėtų padalinti užimamą laiką iš failo dydžio ir apskaičiuoti normalizuotas trukmes.

Atlikę skaičiavimus aiškiai matome, kad įkeliant itin mažus bei mažus (iki 5,25MB dydžio) failus, saugykla užtrunka daugiau nei dvigubai laiko, nei įkeliant didesnius failus. Taip vyksta dėl to, kad operacijos turi tam tikrus pasiruošimo veiksmus, kurių užimama laiko trukmė nepriklauso nuo operacijos sudėtingumo. Šie veiksmai sudaro didesnę dalį bendro operacijos laiko, kai operacija yra

itin trumpa ir tokiu būdų prailgina bendrą laiką. Lygiai tą patį galima pastebėti ir lyginant failo atstatymo normalizuotą užimamą laiko trukmę tiems patiems failams.

Vidutinio dydžio bei dideliems failams (nuo 39,95 MB dydžio) normalizuota trukmė yra labai panaši, bet bendru atveju mažėja, failo dydžiui didėjant. Tai reiškia, kad saugykla, nors ir nežymiai, tačiau efektyviau veikia su didesniais failais (virš 1 GB).



4.11 pav. Normalizuota bendra failų įkėlimo bei atstatymo trukmė, dalinant laiką pagal failo dydį

4.2. Paskirstytos saugyklos failų patikimumo bei vietos efektyvumo tyrimas

Ne visi kompiuteriai ar serveriai arba kitaip – paskirstytos saugyklos mazgai – gali būti pasiekiami tuo momentu, kai failą bandoma parsisiųsti iš saugyklos. Tokiu atveju bus pasiekiamos ne visos koduotos failo dalys. Paskirstytoje saugykloje failų patikimumas užtikrinamas saugojimo pertekliškumu. Tai reiškia, kad naujo failo kėlimo į saugyklą metu yra išsaugomas didesnis kiekis paketų, negu jų reikia, kad atstatyti pradinį failą. Dėl šios priežasties dalis paketų gali būti nepasiekiami ar net prarasti, tačiau pradinį failą atstatyti vis tiek pavyks sėkmingai. Kyla klausimas, kokia dalį paketų realiomis sąlygomis galima prarasti, tačiau vis dar sėkmingai atstatyti failą.

Šiam tyrimui atlikti buvo sukurtas testinis failas „passwords.txt“, kurio dydis 5,25 MB. Jis įkeltas į paskirstytą duomenų saugyklą ir atitinkamai užkoduotas į 401 paketus. Iš 401 paketų, 101 paketas pagamintas pagal pradinį failą, o dar papildomai 300 paketų (pasirinktai) pagaminti kaip papildomi paketai. Visi paketai buvo užšifruojami individualiai.

Kadangi failą be problemų galima atstatyti, kai turimi visi paketai, pagaminti pagal pradinį failą, kiekvieno testo metu simuliuota, kad bent vieno paketo, pagaminto pagal pradinį failą, nepavyko parsisiųsti ir panaudoti failo atstatyme. Tai reiškia, kad kiekvienu atveju iš 101 paketo buvo ištrintas bent vienas paketas.

Atliekant tyrimą, bandyta atstatyti pradinį failą su vis mažiau pasiekiamų failo dalių, pradedant nuo 400 iš 401 dalių ir baigiant tada, kai failo atstatymas nepavyko Eksperimento rezultatai pateikti lentelėje (**4.2 lentelė**).

Failo palyginimui su originaliu buvo naudojama maišos funkcija.

4.2 lentelė. 5,25 MB failo atstatymo testo suvestinė

Viso sugeneruotų paketų	**Iš jų, pasiekiamų paketų**	**Pasiekiamų paketų dalis pagal visus paketus**	**Pasiekiamų paketų bendra užimama vieta diske (MB)**	**Rezultatas**
401	400	0,99	20,98	Failą atstatyti pavyko
401	390	0,99	20,46	Failą atstatyti pavyko
401	380	0,95	19,93	Failą atstatyti pavyko
401	370	0,92	19,41	Failą atstatyti pavyko
<.....>				
401	110	0,27	5,77	Failą atstatyti pavyko
401	100	0,24	5,25	Failo nepavyko atstatyti

Buvo atliktas pakartotinis tyrimas su 110-100 pasiekiamų paketų (**4.3 lentelė**):

4.3 lentelė. Pakartotinio 5,25 MB failo atstatymo testo suvestinė.

Viso sugeneruotų paketų	Iš jų, pasiekiamų paketų	Pasiekiamų paketų dalis pagal visus paketus	Pasiekiamų paketų bendra užimama vieta diske (MB)	Rezultatas
401	110	0,27	5,77	Failą atstatyti pavyko
401	109	0,27	5,72	Failą atstatyti pavyko
401	108	0,26	5,67	Failą atstatyti pavyko
401	107	0,26	5,61	Failą atstatyti pavyko
401	106	0,26	5,56	Failą atstatyti pavyko
401	105	0,26	5,51	Failą atstatyti pavyko
401	104	0,25	5,46	Failą atstatyti pavyko
401	103	0,25	5,40	Failą atstatyti pavyko
401	102	0,25	5,35	Failą atstatyti pavyko
401	101	0,25	5,30	Failą atstatyti pavyko
401	100	0,25	5,25	Failo atstatyti nepavyko

Rezultatas: sėkmingai parsisiųsti failą pavyko turint bent apie 25% failo dalių, kai buvo generuojama 300 papildomų fragmentų ir 101 pradinio failo fragmentas. Visų failo dalių bendra užimama vieta lyginama su pradinio failo užimama vieta yra 100,92%. Taip pat būtina atkreipti dėmesį, kad mažiausias pasiekiamų paketų kiekis, kad sėkmingai būtų atstatytas failas yra 101 paketas – toks pat kiekis paketų, kiek ir buvo sugeneruota pagal pradinį failą (101).

Taip pat buvo palygintas šis rodiklis su situacija, jei į saugyklą įkeltume didesnį failą. Į saugyklą buvo įkeltas 616,22 MB failas ir tyrimas pakartotas. Pagal šį failą buvo sukurta 11941 paketų. Iš jų 2000 papildomi, o 9941 pagrindiniai. Visi jie šifruoti ir saugoti į failą individualiai. Bandytas atstatyti failas po to, kai prarasti, arba, šiuo atvejų ištrinti 1999, 2000 ir 2001 paketų. Rezultatai pateikti lentelėje (**4.4 lentelė**).

4.4 lentelė. 616,22 MB failo atstatymo eksperimento suvestinė

Viso sugeneruotų paketų	Iš jų, pasiekiamų paketų	Pasiekiamų paketų dalis pagal visus paketus	Pasiekiamų paketų bendra užimama vieta diske (MB)	Failo atstatymo rezultatas
11941	9942	0,83	616,33	Failą atstatyti pavyko, maišos reikšmė atitinka
11941	9941	0,83	616,27	Failą atstatyti pavyko, maišos reikšmė atitinka
11941	9940	0,83	616,21	Failo atstatyti nepavyko

Kaip ir buvo galima tikėtis, failui atstatyti dažniausiai pakanka bent vienu paketu daugiau, negu buvo pagaminta pradinių paketų. Šio failo atveju mažiausias reikalingų paketų kiekis užima 616,27 MB ir tai sudaro 100,008% pradinio failo dydžio. Tai reiškia, kad šis dydis priklauso tik nuo vieno paketo dydžio, o ne nuo bendro failo dydžio, tačiau tiek vienu, tiek kitu atveju reikalingas pertekliškumo laipsnis yra itin mažas, mažesnis nei 1%.

Pakartotas testas suliejant gautus paketus ir šifruojant juos į failų grupes, taip siekiant sumažinti didžiulį failų kiekį.

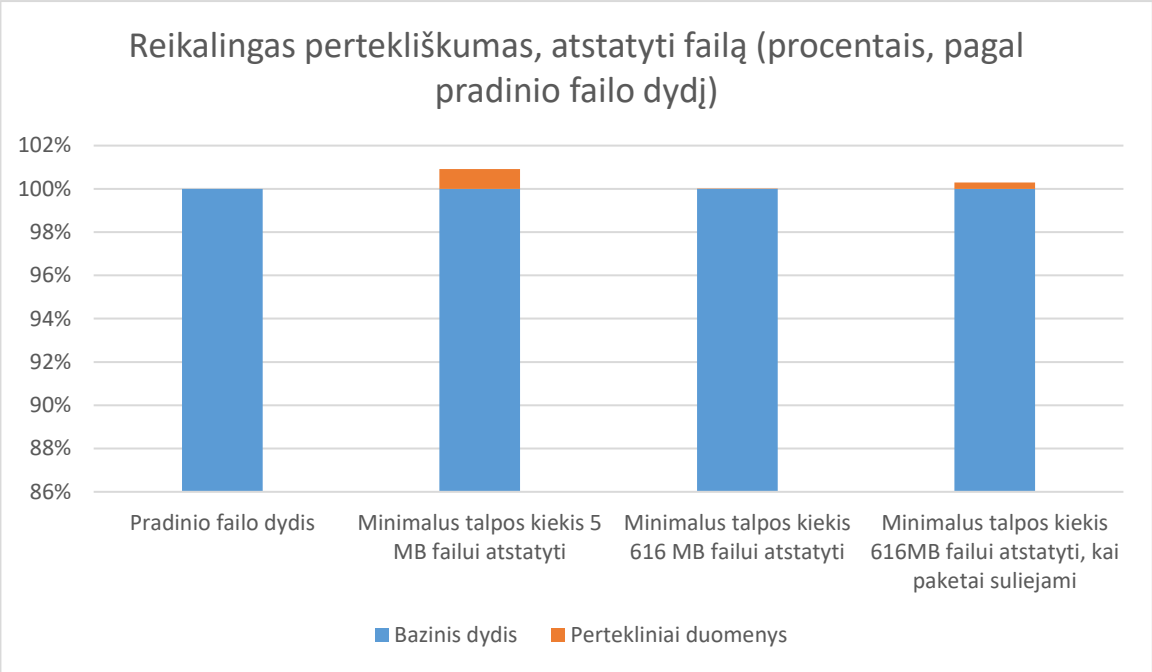
Naudotas tas pats 616,22 MB failas, tačiau gauti paketai sulieti į failus po 82 paketus. Testo metu vieno paketo dydis buvo 65 KB (nešifruoto), dėl to paketai sulieti į failus gaminant 5,34 MB (arba mažesnius) failus. Po šifravimo failo dydžiai padidėjo, tačiau vis vien tokiu atveju gautas mažesnis bendras failų kiekis, nors paketų kiekis liko nepakitęs. Testo metu buvo ištrintas tam tikras failų kiekis ir bandytas atstatyti pradinis failas. Rezultatai pateikti lentelėje (**4.5 lentelė**).

4.5 lentelė. 616,22 MB failo atstatymo, kai paketai suliejami, eksperimento suvestinė

Viso sugeneruotų paketų	Viso sugeneruota failų	Iš jų, pasiekiamų failų	Iš jų, pasiekiamų paketų	Pasiekiamų paketų dalis pagal visus paketus	Pasiekiamų paketų bendra užimama vieta diske (MB)	Rezultatas
11941	146	122	9973	0,83	618,25	Failą atstatyti pavyko
11941	146	121	9891	0,82	613,17	Failo atstatyti nepavyko

Failą atstatyti pavyko, kol pasiekiami buvo bent 9973 paketai iš 122 failų. Praradus dar vieną failą, atstatyti failo nebepavyko. 122 failai bendrai užėmė 618,25 MB disko vietos, o tai sudaro 100,3% pradinio failo dydžio.

Tai reiškia, kad paketų suliejimas šiek tiek padidino pertekliškumo laipsnį, nes vieno failo praradimo momentu, iškart prarandami 82 paketai, tačiau toks pertekliškumas vis dar yra labai geras rezultatas (žr. **4.12 pav.**).



4.12 pav. Sukurto metodo pertekliškumo tyrimo rezultatai, skirtingų naudojimo scenarijų metu

4.3. Ribinio atstatymo galimybės tyrimas

Praeitame tyrime buvo nustatytas įprastas minimalus failo fragmentų kiekis, reikalingas sėkmingam failo atstatymui, tačiau analizės dalyje buvo nustatyta, kad dėl *Raptor* kodų ribojimo, šis kiekis nėra absoliutus. Kadangi generuojami paketai nėra deterministiniai, teoriškai tam tikrais atvejais net ir turint daugiau N ar N + 1 paketų, kur N yra paketai, pagaminti pagal pradinį failą, atstatyti failą nėra galimybės.

Atliekame testavimą šiai situacijai ištirti. Į saugyklą įkėlėme 5,00 MB failą. Fragmento dydis nustatytas 1400 B, fragmentų pagal pradinį failą sukurta 3572. Skirtingų testų metu nustatėme mažą, vidutinį bei didelį pertekliškumą (10, 1000 bei 100000 papildomų fragmentų). Ištrynėme kritinį fragmentų kiekį atsitiktiniu būdu. Tai yra N + 1, N, N – 1, N – 2 kiekį, kur N yra papildomų fragmentų kiekis. Testą kartojome didelį kiekį kartų, kad būtų galima tiksliau apskaičiuoti tikimybę sėkmingai atstatyti failą.

Testavimo rezultatai pateikti **4.6 lentelėje**.

4.6 lentelė. Failo atstatymo tikimybė turint ribinį kiekį fragmentų

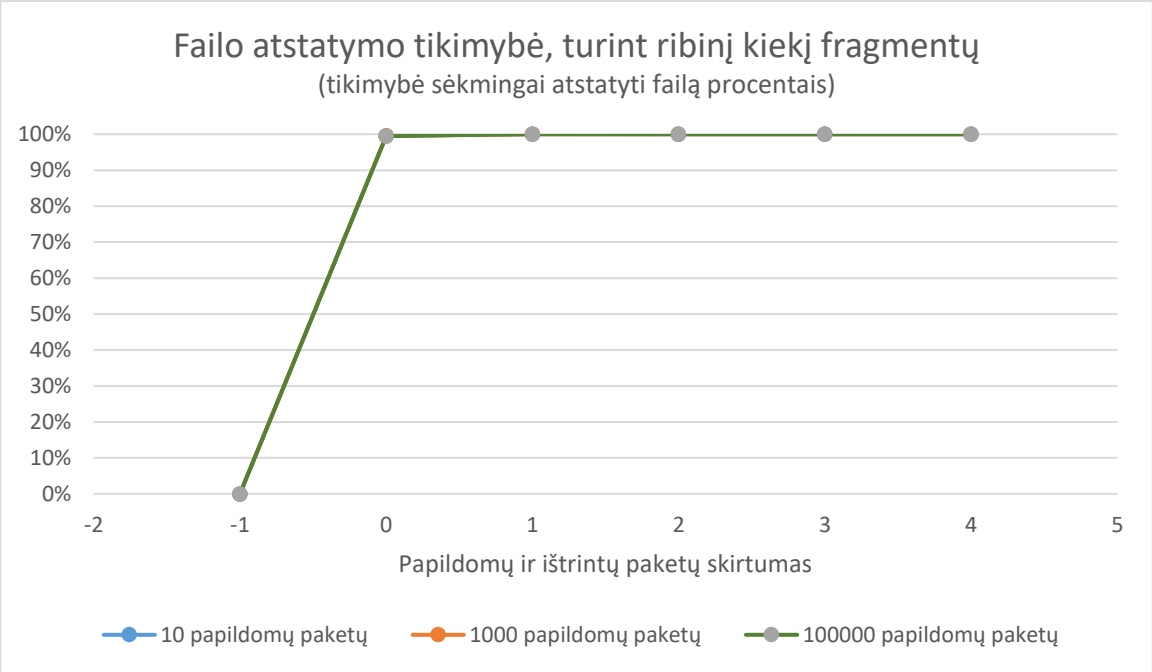
Pradinis fragmentų kiekis	**Papildomų fragmentų kiekis**	**Ištrintų fragmentų kiekis**	**Nepavykusių atstatyti testų kiekis**	**Visų testų kiekis**	**Tikimybė sėkmingai atstatyti (%)**
3572	10	11	25000	25000	0
3572	10	10	77	16000	99,5351
3572	10	9	1	115000	99,9991
3572	10	8	0	70000	100
3572	1000	1001	25000	25000	0
3572	1000	1000	105	23000	99,5459
3572	1000	999	2	62000	99,9968
3572	1000	998	0	100000	100
3572	1000000	1000001	25000	25000	0
3572	1000000	1000000	62	11480	99,4599
3572	1000000	99999	3	34000	99,9913
3572	1000000	99998	0	25000	100

Pagal lentelės duomenis matyti, kad nepriklausomai nuo sugeneruotų papildomų paketų kiekio, jei ištrintų paketų kiekis yra nors truputį didesnis, negu papildomų paketų kiekis, failo atkurti galimybės nėra (situacija N – 1).

Kai papildomų paketų kiekis sutampa su ištrintų paketų kiekiu (ištrintų paketų kiekis yra lygus N), tikimybė sėkmingai atkurti failą siekia apie 99,5%.

Kai po atsitiktinio trynimo turime vienu paketu daugiau, negu pradinių paketų kiekis (N + 1), tikimybė atkurti failą yra beveik 100% arba apie 99,99%

Jei po atsitiktinio trynimo turime bent du papildomus paketus (N + 2), tikimybė atstatyti failą yra 100% (pagal 180 tūkst. bandymų).

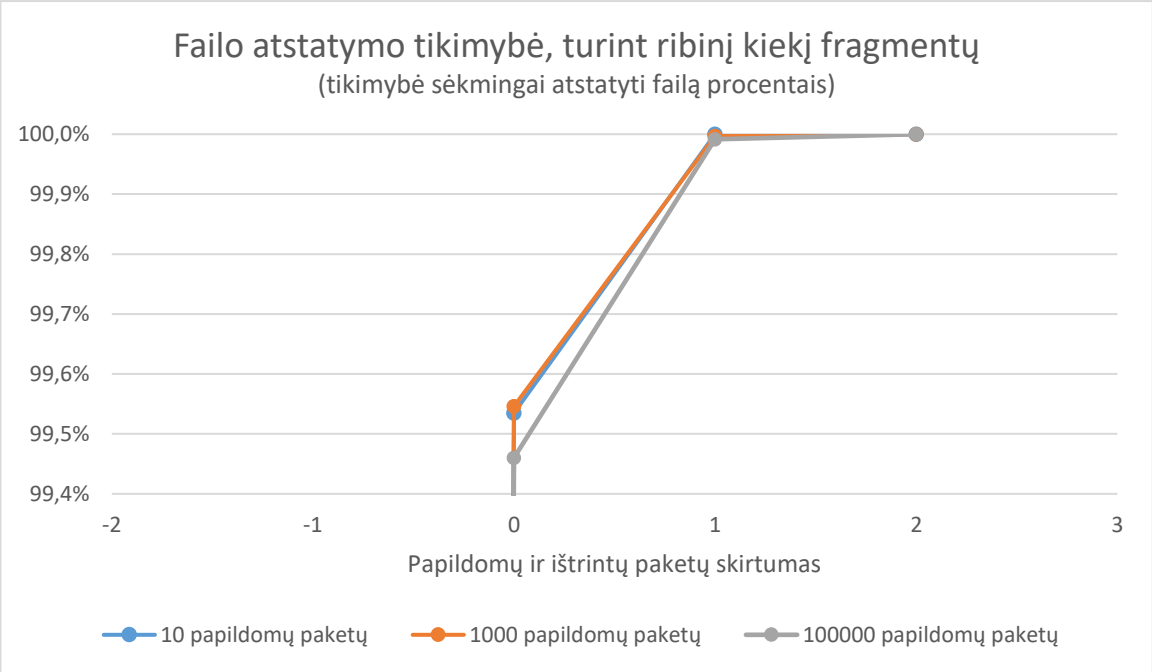


4.13 pav. Failo atstatymo tikimybės priklausomybė nuo papildomų ir ištrintų paketų skirtumo

Taipogi galima palyginti, ar skiriasi tikimybė atkurti pradinį failą, turint ribinį kiekį paketų, nuo bendro sugeneruotų paketų skaičiaus. Vienu atveju papildomai sugeneruota 10 paketų, antru – 1000 paketų, o trečiu 100000 papildomų paketų.

Kadangi tikimybės labai panašios, pateiktas stambaus mastelio grafikas, kuriame matomos tikimybės tarp 99,4% ir 100%.

Tikimybė atstatyti failą, kai turime N kiekį paketų yra beveik identiška ir skiriasi tik per 0,06%, nors paketų kiekis skiriasi ir 100 kartų, todėl galima teigti, kad tikimybė atstatyti failą ribinėje situacijoje nuo pradinių sugeneruotų paketų kiekio nepriklauso.



4.14 pav. Failo atstatymo tikimybės priklausomybė nuo papildomų ir ištrintų paketų skirtumo (ties 99,4% - 100% tikimybe)

4.4. Tyrimo apibrendrinimas

1. Atlikus eksperimentus įkeliant ir parsisiunčiant skirtingo dydžio failus iš saugyklos bei įvertinus statistinį interneto ryšio greitį galime daryti prielaidą, kad įkeliant ar parsisiunčiant didesnio dydžio failus didžiausią dalį sudaro failo dalių siuntimas tinklu.
2. Pirmoje vietoje sąraše pagal laiko trukmę tiek įkeliant, tiek parsisiunčiant mažo dydžio failus buvo metaduomenų saugojimo operacija, tačiau visai nedaug atsilieka ir šifravimo operacija.
3. Eksperimentas įrodo, kad saugyklos vientisumas ir patikimumas atitinka *RaptorQ* specifikaciją ir su dideliu pasitikėjimo laipsniu leidžia atstatyti pradinį failą turint bent N + 1 paketą, kur N – pradiniai failo paketai, nepriklausomai nuo to, kurie paketai buvo prarasti.
4. Saugyklos pertekliškumo laipsnis yra itin mažas ir pakanka turėti mažiau nei 101% pradinio failo paketų, kad pavyktų atstatyti failą su didesne nei 99% tikimybe. Šis dydis nuo sugeneruotų paketų kiekio nepriklauso.

Išvados

1. Analizės skyriuje buvo išgrynintos pagrindinės duomenų saugumo savybės, duomenų vientisumas, patikimumas bei konfidencialumas. Išanalizavus šaltinius, buvo nuspręsta, kad kuriama sistema užtikrins duomenų vientisumą bei patikimumą fontaniniais klaidų taisymo kodais bei maišos funkcijų reikšmėmis. Duomenų konfidencialumas bus užtikrintas naudojantis kriptografija, tiek simetrinėmis, tiek asimetrinėmis šifravimo sistemomis.
2. Atlikus panašaus principo paskirstytų saugyklų paiešką, nebuvo rasta sistema, kuri atitiktų visus iškeltus reikalavimus. Dauguma jų neturėjo galimybės už saugomą informacijos kiekį atsiskaityti savo disko vieta, neturėjo galimybės reitinguoti vartotojus arba užtikrinti failų konfidencialumą.
3. Sprendimo projektavimo etape buvo pasiūlytas metodas duomenims saugoti paskirstytoje duomenų saugykloje. Siūlomas metodas šifruotus vartotojų duomenis saugo su pertekliškumu, kurį užtikrina kodavimui naudojamas fontaninis klaidų taisymo kodas – *Raptor*. Toks kodavimas leidžia failus padalinti į fragmentus ir atstatyti failą turint tik dalį fragmentų, nepaisant jų eiliškumo. Sprendimas apima ir papildomas funkcijas, tokias kaip vartotojų naudingumo įverčio taikymas, kuris leidžia turėti efektyvesnius paskirstytos saugyklos mazgus, vartotojo diske esančių failų kontrolė bei specialus failo parsisiuntimo mechanizmas, skirtas atstatyti failą net ir tada, kai didžioji dalis mazgų yra nepasiekiami.
4. Atlikus prototipo realizacijos etapą buvo sėkmingai realizuotas efektyvaus saugojimo paskirstytoje saugykloje prototipas, kuris buvo sukurtas naudojantis *Docker* aplinka bei *Python* programavimo kalba. Sukurtas prototipas leidžia įkelti bei parsisiųsti failą iš saugyklos, simuliuoti fragmentų praradimus, sulieti fragmentus į grupes, testuoti failo atstatymo tikimybes.
5. Atlikus prototipo kiekybinį tyrimą buvo nustatyta, kad saugykla veikia su įvairaus dydžio failais, ir normalizuota greitaveika pagal failo dydį yra daugmaž vienoda ir siekia 50 sekundžių apdoroti vienam gigabaitui duomenų. Taip pat nustatyta, kad didesniems nei 50 MB failams sistemos našumo ribojantis faktorius buvo tinklo greitaveika, o saugyklos operacijos sudaro mažiau nei 50% laiko tiek įkeliant, tiek ir parsisiunčiant failus.
6. Atlikus prototipo kokybinį tyrimą buvo nustatyta, kad saugojimo pertekliškumas yra itin mažas ir kad pradiniam failui atkurti su bent 99% pasikliautinumo laipsniu pakanka turėti tik vos daugiau nei 100% pradinio failo fragmentų, o turint vos vienu fragmentu daugiau, tikimybė sėkmingai atstatyti failą praktiškai tampa 100%.

Literatūros sąrašas

1 . S. Samonas, „The CIA strikes back: redefining confidenciality, integrity and availability in security,“ *JISSec Journal of Information System Security,* pp. 21-45, 2014.

2 . S. Mirrezaei, „Towards systematic Luby transform codes: optimisation design over binary erasure channel,“ t. 56, nr. 11, pp. 550-553, 2020.

3. S. A., „Raptor codes,“ *IEEE Transactions on Information Theory,* pp. 2551-2567, 2006.

4. Q. I. A. S. M. Luby, „RaptorQ Forward Error Correction Scheme for Object Delivery,“ IETF, 2011.

5. Qualcomm, „RaptorQ™ Technical Overview,“ Qualcomm, 2010.

6. S. G. V. M. Joshua Brakensiek, „Generic Reed-Solomon codes achieve list-decoding capacity,“ *arXiv,* 2023.

7. L.-J. Saiz-Adalid, J. Gracia-Morán ir D. Gil-Tomás, „Reducing the Overhead of BCH Codes: New Double Error Correction Codes,“ t. 9, nr. 11, p. 1897, 2020.

8. V. S. Shetty, „A Survey on Performance Analysis of Block Cipher,“ *Proceedings of the Fifth International Conference on Inventive Computation Technologies* , pp. 167-174, 2020.

9. O. O. S. M. P. B. Bos Jopee W., „Efficient Hashing Using the AES Instruction Set,“ *Cryptographic Hardware and Embedded Systems – CHES 2011,* pp. 507-522, 2011.

10. B. Schneier, „Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish),“ *Lecture Notes in Computer Science,* pp. 191-204, 1994.

11. J. K. D. W. Bruce Schneier, „The Twofish Team’s Final Comments on AES,“ 2000.

12. E. B. L. K. Ross Anderson, „Serpent: A Proposal for the Advanced Encryption Standard,“ 2000.

13. E. Barker, „Recommendation for Key Management,“ National Institute of Standards and Technology, 2020.

14. F. Maqsood, „Cryptography: A Comparative Analysis for Modern,“ *International Journal of Advanced Computer Science and Applications,* pp. 442-448, 2017.

15. E. C. Q. D. I. P. Saez Yago, „Evolutionary hash functions for specific domains,“ *Applied Soft Computing,* t. 78, pp. 58-69, 2019.

16. CrowdSource, „FAQ - Polycloud,“ 2020. [Tinkle]. Available: https://crowdstorage.com/faq/. [Kreiptasi 15 Sausio 2021].

Informacijos šaltinių sąrašas

1. https://osp.stat.gov.lt/skaitmenine-ekonomika-ir-visuomene-lietuvoje-2020/zmones-ir-verslas-internete (žiūrėta 2022-05-01)