VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS INSTITUTAS

Formalių specifikacijų taikymas projektuojant išskirstytas sistemas

Applying Formal Specifications to Design Distributed Systems

Magistro darbo planas

Atliko:	Matas Savickis	(parašas)
Darbo vadovas:	Karolis Petrauskas, Doc., Dr.	(parašas)
Recenzentas:	Valaitis Vytautas, Asist., Dr.	(parašas)

Vilnius – 2021

TURINYS

Įvadas

Šiais laikais kai kurios programų sistemos yra išskirstytos [Smi18]. Tokios sistemos, kaip sufleruoja pavadinimas, yra kuriamos išskirstant skaičiavimo mazgus į atskiras, savarankiškas dalis [CDK05]. Ne kaip monolitinės sistemos, išskirstytos sistemos gali veikti skirtingose serverinėse, kurios gali būti įvairiose geografinėse vietose [Shi06]. Toks sistemos išskirstymas pasižymi šiomis savybėmis:

1. Pasiekiamumas (angl. *Availability*) – vartotojas gali pasiekti sistemą bet kuriuo metu [GLA+13].

2. Patvarumas (angl. *Durability*) – išskirstytos sistemos užtikrina duomenų išlaikymą ir pastovų veikimą net jeigu ir vienas iš paskirstytos sistemos mazgų nustotų veikti dėl sistemos sutrikimų, sukeltų programos klaidų arba gamtos katastrofų, tokių kaip: gaisrai, potvyniai ir panašios nelaimės. Ši savybė taip pat užtikrina sistemos gebėjimą atsistatyti ir neprarasti duomenų po minėtų įvykių [RP10].

3. Plečiamumas (angl. *Scalability*) – didėjant vartotojų skaičiui bei programų sistemos kompleksiškumui išskirstytos sistemos užtikriną vertikalų (padidinti mazgo techninės įrangos galingumą) bei horizontalų (padidinti mazgų skaičių sistemoje) plečiamumą [JW00].

4. Našumas (angl. *Efficiency*) – sistemos vartotojų skaičius dažniausiai būna nepastovus, jis kinta dienos metu arba atitinkamais metų periodais. Išskirstytos sistemos padeda užtikrinti našų įrangos resursų naudojimą sumažinant įrangos galingumą (kai vartotojų skaičius yra nedidelis) bei padidinant galingumą (kai sistemos apkrova padidėja).

Šiuo metu yra sukurta keletas atviro kodo išskirstytų sistemų, padedančių apdoroti realaus laiko duomenis. Viena iš tokių išskirstytų sistemų yra Apache Kafka (toliau – Kafka) [20b].

Kafka buvo pradėta kurti kompanijos LinkedIn [20b]. 2012 metais Kafka sistema buvo perduota Apache Software Foundation tolesniam vystymui. Šiuo metu Kafka platforma yra žinučių siuntimo sistema, kuri pasižymi lengvu plečiamumu, patvarumu, patikimumu ir greičiu. Duomenys Kafka platformoje yra išsaugomi saugiu, trukdžiams atspariu būdu. Kafka kūrėjų teigimu, šiuo metu platformą naudoja daugiau negu 80 procentų didžiausių Jungtinių Amerikos Valstijų įmonių [20b]. Kafka platforma yra plačiai naudojama įvairiose srityse, tokiose kaip: žurnalistika, debesijos paslaugos, muzikos srauto paslaugos, telekomunikacijos, bankinės paslaugos ir daugelis kitų [20b].

Norėdami užtikrinti Kafka platformos kokybę, kūrėjai yra įgyvendinę skirtingų testų [20c]. Testai padeda atskleisti programos klaidas arba pasakyti ar naujas kodas nepaveikė seniau parašyto funkcionalumo [Whi00]. Tačiau net ir laikantis geriųjų testavimo praktikų nepavyksta išvengti programos klaidų. Net ir paskyrus daugiau resursų testavimui sudėtinguose sistemose, tokiose kaip Kafka, pilnas sistemos testavimas yra neįmanomas [SYC+04]. Todėl norint atrasti subtilesnius sisteminius sutrikimus tenka naudoti kitus metodus, tokius kaip formalus specifikavimas ir verifikavimas.

Formalios specifikacijos yra matematinės technikos skirtos apibūdinti sistemų elgseną ir padėti kuriant jų dizainą naudojant griežtas ir veiksmingas priemones [HP95]. Turint sistemos formalią specifikaciją galima ja pasinaudoti vykdant formalų verifikavimą ir parodant, kad algoritmas yra adekvatus pagal sukurtą specifikaciją. Sudarinėti formalią sistemos specifikaciją galima ir nepradėjus įgyvendinti sistemos, turint tik jos dizainą. Formaliai verifikuota specifikacija suteikia informacijos apie dizaino korektiškumą ir įgalina objektyviai koreguoti sistemos dizainą dar prieš pradedant jį įgyvendinti. Formalios specifikacijos sudaromos pasinaudojant tam tikromis kalbomis arba įrankiais. Viena iš tokių formalaus specifikavimo kalbų yra TLA$^+$ [Lam02].

TLA$^+$ yra formalios specifikacijos kalba, kurią sukūrė Leslie Lamport [Lam02]. Leslie Lamport 1980 metais sukūrė laiko veiksmų logiką (angl. *Temporal Logic of Actions*) [Lam94] pasinaudodamas Amir Pnueli 1977 metais sukurta laiko logika (angl. *Temporal Logic*) [Pnu77]. 1999 metais Leslie Lamport naudodamasis laiko veiksmų logika sukūrė formalaus specifikavimo kalbą TLA$^+$ [Lam02].

TLA$^+$ kalba yra skirta kurti konkurencinių ir išskirstytų sistemų formalias specifikacijas ir jas verifikuoti. Naudojant TLA$^+$ galima specifikuoti šias išskirstytų sistemų savybes [Lam19]:

1. Gyvumas – geri dalykai įvyksta programos vykdymo metu. Sistema galiausiai atliks jai paskirtą užduotį arba pateks į norimą būseną.

2. Saugumas – blogi dalykai neatsitiks programos vykdymo metu. Sistema nesustos veikti dėl netikėtai iškilusios klaidos.

Kadangi TLA$^+$ specifikacijos yra rašomos formalia kalba, tai leidžia patikrinti sukurtos specifikacijos saugumo ir gyvumo savybes.

Šias savybes mes galime patikrinti naudodamiesi TLC Model Checker (modelio tikrintoju). TLC yra išreikštinės būsenos (explicit-state) modelio tikrintojas, kurio paskirtis yra palaipsniui pasiekti visas galimas sistemos būsenas pagal nurodytą formalią specifikaciją [YML99]. Tačiau kartais pagal sukurtą formalią specifikaciją susidaro labai daug būsenų, kurias sistema gali pasiekti, todėl tampa nepraktiška naudoti TLC. Tokiu atveju galime naudotis TLA$^+$ specifikacijos įrodymo sistema TLAPS [CDL$^+$12]. TLAPS yra įrodymų sistema skirta patikrinti TLA$^+$ įrodymus. Šios sistemos paskirtis yra patikrinti pateiktus teoremų įrodymus. Įrodžius teoremą laikoma, kad TLA$^+$ specifikacija yra korektiška.

Yra ir kitų formalaus specifikavimo kalbų kurias galėtume naudoti šiame darbe. Viena iš jų Z formalaus specifikavimo kalba [ORe17], kuri sėkmingai buvo naudota specifikuoti UNIX failų sistemai [Bow96], bei Oxfordo universiteto paskirstytų skaičiavimų projekte [Bow96]. Dar viena formalaus specifikavimo kalba yra VDA [BC$^+$78], kuria buvo specifikuoti bendros atminties sinchronizavimo algoritmai [SVH98]. Tačiau šiam darbui buvo pasirinkta TLA$^+$ kalba dėl jos pritaikymo išskirstytoms sistemoms naudojant būsenų mašiną [Lam02], esamų sėkimingo pritaikymo pavyzdžių specifikuojant išskirstytas sistemas [18; Gus04; JHS20; NRZ$^+$14] bei gausaus TLA$^+$ įrankių pasirinkimo.

Tačiau parašyti formalią specifikaciją yra negana. Kartais gali nutikti taip, kad algoritmo realizacija neatitiks jos reikalavimų. Mūsų sudaryta specifikacija gali būti adekvati pagal pateiktus reikalavimus, tačiau sistemos kūrėjai šiuos reikalavimus gali įgyvendinti nekorektiškai. Norint

to išvengti turime patikrinti ar sistemos įgyvendinimas atitinka specifikaciją. Tą pasieksime atlikdami formalią verifikaciją naudodami Kafka įvykių žurnalą. Tam atlikti pasinaudosime modeliu paremtu pėdsakų tikrinimo (angl. *Model-Based Trace-Checking*) metodu [20f]. Metode aprašomi šie žingsniai:

1. Įgyvendinti programą ir paprastus testus.
2. Pridėti kodą, kuris sektų programos būseną ir įrašytų ją į failą.
3. Sukurti formalią specifikaciją parašytam kodui.
4. Paleisti sistemos būsenos failą per įrankius skirtus patikrinti ar sistemos būsena, failo duomenys yra adekvatūs pagal formalią specifikaciją.

Šiame darbe pirmą žingsnį praleisime, nes specifikuosime jau įgyvendintus algoritmus Kafka platformoje. Antrame žingsnyje pridėsime kodą, kuris registruos sistemos būseną ir įrašinės ją į tekstinius failus. Trečiame žingsnyje sukursime formalią specifikaciją naudodamiesi TLA$^+$. Ketvirtame žingsnyje, naudodami TLC, patikrinsime ar sistemos būsenos failo duomenys yra adekvatūs pagal sukurtą formalią specifikaciją. Apdorosime būsenos duomenis ir pasinaudodami TLC tikrinsime ar tokią būseną galima pasiekti pagal mūsų sukurtą specifikaciją.

Temos aktualumas bei naujumas

Viena iš formalių specifikacijų ir TLA$^+$ panaudojimo industrijoje sėkmės istorijų yra Amazon Web Service (AWS) komandos 2014 metais išleistas straipsnis [NRZ$^+$14]. Straipsnyje rašoma, kad AWS komanda naudojo TLA$^+$ sudarant formalias specifikacijas dešimtyje projektų. Tuo metu AWS turėjo 7 komandas, kurios naudojosi TLA$^+$ kurdamos naujas programų sistemas. AWS sistemos specifikavimo metu buvo surasta 10 iki šiol neatrastų sisteminių klaidų, kurių atradimas ir pasiūlyti ištaisymai atskleidė tolimesnes sistemos klaidas, kurios taip pat buvo ištaisytos. Straipsnyje įvardinta ir kita, netiesioginė nauda gauta formaliai specifikuojant sistemas: pagerėjęs bendras sistemos suvokimas, padidėjęs produktyvumas ir inovacijos.

Dar viena sėkmės istorija yra 2018 metais Kafka Summit konferencijoje pristatyta Kafka TLA$^+$ formali specifikacija, kurią sukūrė Jason Gustafson [Gus04]. Pristatyme buvo parodyta kaip pritaikant TLA$^+$ bei specifikuojant Kafka duomenų replikavimo algoritmą buvo surastos ir pataisytos 3 retai atsitinkančios programos klaidos. Taisant taip pat rastos ir pataisytos dar kelios klaidos.

Šiame darbe specifikuosime ir verifikuosime Kafka kūrėjų pasiūlytą Raft algoritmo [HSM$^+$15] realizaciją [20a]. Raft protokolas yra skirtas pasiekti susitarimą išskirstytoje sistemoje. Norint pasiekti susitarimą tarp sistemos mazgų, kiekvienas mazgas turi turėti lyderio arba sekėjo rolę. Algoritme lyderis yra atsakingas už informacijos replikavimą savo sekėjams. Lyderis tam tikru metu praneša sekėjams apie savo egzistavimą. Jeigu sekėjas nesulaukia signalo iš lyderio, sistemoje prasideda naujo lyderio rinkimas.

Kafka pasiūlyto Raft algoritmo realizacija būtų naudojama duomenų replikavimui [20a]. Nors Raft algoritmui jau yra sukurta formalių specifikacijų TLA$^+$ kalba [18], tačiau pasiūlyta realizacija skiriasi nuo iki šiol sukurtų specifikacijų, todėl reiktų sukurti atskirą formalią specifikaciją bei ją verifikuoti.

Kafka sutrikimų sekimo sistemoje yra aprašoma ir kitų Kafka sutrikimų [20e] , kuriuos būtų galima formaliai specifikuoti ir verifikuoti taip parodant algoritmų problemas. Ši informacija galėtų padėti sistemos kūrėjams ištaisyti Kafka sutrikimus. Sutrikimai, kuriuos būtų galima formaliai specifikuoti ir verifikuoti:

- Kafka Valdiklis neišsijungia teisingai kai įvyksta techninis gedimas [War20]
- Lenktynių sąlyga klasėje FindCoordinatorFuture visam laikui nutraukia sąsaja su grupės koordinatoriumi [20h]
- Lengtyvių sąlyga gali sukelti atsilikimą kitoms aktyvioms užduotims [20g]

Visi šie pateikti sutrikimai buvo įvardinti kaip kritiniai ir nepradėti taisyti, todėl papildoma informacija gauta formaliai specifikuojant juos galimai padėtu išspręsti šias problemas.

Iki šiol, kiek yra žinoma, Kafka platforma neakademiniame kontekste buvo specifikuota tik vieną kartą [Gus04], o sukurta specifikacija atnešė naudos padedant surasti sistemos klaidas. Panašią mokslininkų sėkmę matome ir Amazon Web Service formalios specifikacijos sudarymo tyrimuose [NRZ$^+$14]. Dėl papildomų Kafka formalių specifikacijų stokos ir praeityje pasisekusio formalaus specifikavimo išskirstytuose sistemose manome, kad papildomi tyrimai Kafka platformoje atneštų naudos surandant algoritmų klaidas arba užtikrinant, kad specifikuotose algoritmuose jų nėra. Šiuo metu Kafka sisteminių klaidų registre [20d] yra išspręstų ir neišspręstų klaidų, kurių verifikavimas padėtų atskleisti naujas klaidas arba įrodytų, kad klaidos ištaisytos adekvačiai. Kafka platforma turi daug naudotojų [20b], todėl tolimesnis kokybės užtikrinimas Kafka platformoje atneštų naudą.

Kurti specifikacijas Kafka platformose naudojamiems algoritmams gali būti naudinga ir didesnei aibei sistemų. Sėkmingai specifikavus algoritmus, naudojamus Kafka platformoje, būtų galima įrodyti adekvatumą daug didesnei išskirstytų sistemų aibei, kurioje yra naudojami tokie pat algoritmai. Šiuo metu yra straipsnių, kuriuose formaliai verifikuojami išskirstytų sistemų algoritmai [Lam05]. Jie yra naudojami kurti išskirstytas sistemas, todėl yra tikimasi, kad panašius rezultatus pavyks pasiekti specifikuojant Kafka platformos algoritmus.

Darbo tikslas

Parodyti Apache Kafka algoritmų korektiškumą naudojantis formaliu specifikavimu, bei įvardinti problemas specifikuotose algoritmuose.

Uždaviniai

1. Formaliai specifikuoti pasirinktus Kafka platformos algoritmus naudojant TLA$^+$ specifikavimo kalbą. Šiame žingsnyje formaliai aprašysme pasirinktus algoritmus ir jų savybes, kad galėtume patikrinti jų korektiškumą.

2. Verifikuoti ar pagal sukurtą specifikaciją Kafka platforma veikia korektiškai. Verifikacija bus atliekama naudojant modeliu paremtu pėdsakų tikrinimo (angl. *Model-Based Trace-Checking*) metodu.

3. Esant poreikiui įrodyti specifikacijos savybes naudojant TLAPS. Šios užduoties prireik, jeigu formalios specifikacijos tikrinimas TLC modelio tikrintoju užtruktų perdaug laiko.

4. Surasti kitas paskirstytas sistemas, kuriose yra naudojami šiame darbe formaliai specifikuoti algoritmai. Šiuo uždaviniu parodytume, kad šis darbas būtų pritaikomas didesniai aibei paskirstytų sistemų, ne tik Apache Kafka.

Laukiami rezultatai

1. Pasirinktų Kafka algoritmų specifikacija.

2. Įrodymas apie specifikacijos adekvatumą.

3. Kafka specifikacijos ir įgyvendinimo sutapimo įvertinimas.

4. Išskirti ir specifikuoti išskirstytų sistemų šablonai taikomi kitose platformose.

Literatūra

[18] Raft formal specification with tla+, 2018. URL: https://github.com/ongardie/raft.tla/blob/master/raft.tla.

[20a] A raft protocol for the metadata quorum, 2020. URL: https://cwiki.apache.org/confluence/display/KAFKA/KIP-595%5C%3A+A+Raft+Protocol+for+the+Metadata+Quorum.

[20b] Apache kafka, 2020. URL: https://kafka.apache.org/.

[20c] Apache kafka mirror tests, 2020. URL: https://github.com/confluentinc/kafka/tree/master/tests.

[20d] Kafka - - asf jira, 2020. URL: https://issues.apache.org/jira/projects/KAFKA/issues/KAFKA-10635?filter=allopenissues.

[20e] Kafka sutrikimų registravimo sistema, 2020. URL: https://issues.apache.org/jira/projects/KAFKA.

[20f] Latex is cool, 2020. URL: https://kafka.apache.org/.

[20g] Lengtyvių sąlyga gali sukelti atsilikimą kitoms aktyvioms užduotims, 2020. URL: https://issues.apache.org/jira/browse/KAFKA-9846.

[20h] Lengtyvių sąlygos sutrikimas findcoordinatorfuture klasėje, 2020. URL: https://issues.apache.org/jira/browse/KAFKA-10793.

[BC+78] Dines Bjorner, JONES CB ir k.t. The vienna development method: the meta-language. 1978.

[Bow96] Jonathan Peter Bowen. *Formal specification and documentation using Z: A case study approach*, tom. 66. International Thomson Computer Press London, 1996.

[CDK05] George F Coulouris, Jean Dollimore ir Tim Kindberg. *Distributed systems: concepts and design*. pearson education, 2005.

[CDL+12] Denis Cousineau, Damien Doligez, Leslie Lamport, Stephan Merz, Daniel Ricketts ir Hernán Vanzetto. Tla+ proofs. *International Symposium on Formal Methods*, p. 147–154. Springer, 2012.

[GLA+13] Trinabh Gupta, Joshua B. Leners, Marcos K. Aguilera ir Michael Walfish. Improving availability in distributed systems with failure informers. *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, p. 427–441, Lombard, IL. USENIX Association, 2013-04. ISBN: 978-1-931971-00-3. URL: https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/leners.

[Gus04] Jason Gustafson. Hardening kafka replication. https://kafka-summit.org/sessions/hardening-kafka-replication, 2004. Pristatymas konferencijoje.

[HP95] Gerard J Holzmann ir Doron Peled. An improvement in formal verification. *Formal Description Techniques VII*, p. 197–211. Springer, 1995.

[HSM+15] Heidi Howard, Malte Schwarzkopf, Anil Madhavapeddy ir Jon Crowcroft. Raft refloated: do we have consensus? *SIGOPS Oper. Syst. Rev.*, 49(1):12–21, 2015-01. ISSN: 0163-5980. DOI: 10.1145/2723872.2723876. URL: https://doi.org/10.1145/2723872.2723876.

[YML99] Yuan Yu, Panagiotis Manolios ir Leslie Lamport. Model checking tla+ specifications. *Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, p. 54–66. Springer, 1999.

[JHS20] A Jesse Jiryu Davis, Max Hirschhorn ir Judah Schvimer. Extreme modelling in practice. *arXiv e-prints*:arXiv–2006, 2020.

[JW00] P. Jogalekar ir M. Woodside. Evaluating the scalability of distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 11(6):589–603, 2000. DOI: 10.1109/71.862209.

[Lam02] Leslie Lamport. *Specifying systems*, tom. 388. Addison-Wesley Boston, 2002.

[Lam05] Leslie Lamport. Generalized consensus and paxos, 2005.

[Lam19] Leslie Lamport. Safety, liveness, and fairness, 2019.

[Lam94] Leslie Lamport. The temporal logic of actions. *ACM Trans. Program. Lang. Syst.*, 16(3):872–923, 1994-05. ISSN: 0164-0925. DOI: 10.1145/177492.177726. URL: https://doi.org/10.1145/177492.177726.

[NRZ+14] Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker ir Michael Deardeuff. Use of formal methods at amazon web services, 2014. URL: http://research.%20microsoft.com/en-us/um/people/lamport/tla/formal-methods-amazon.pdf.

[ORe17] Gerard O'Regan. *Z formal specification language*. *Concise Guide to Formal Methods: Theory, Fundamentals and Industry Applications*. Springer International Publishing, Cham, 2017, p. 155–171. ISBN: 978-3-319-64021-1. DOI: 10.1007/978-3-319-64021-1_8. URL: https://doi.org/10.1007/978-3-319-64021-1_8.

[Pnu77] A. Pnueli. The temporal logic of programs. *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, p. 46–57, 1977. DOI: 10.1109/SFCS.1977.32.

[RP10] S. Ramabhadran ir J. Pasquale. Analysis of durability in replicated distributed storage systems. *2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS)*, p. 1–12, 2010. DOI: 10.1109/IPDPS.2010.5470366.

[Shi06] Kenneth W Shirriff. Method and system for establishing a quorum for a geographically distributed cluster of computers, 2006-3 21. US Patent 7,016,946.

[SYC+04] Kevin Sullivan, Jinlin Yang, David Coppit, Sarfraz Khurshid ir Daniel Jackson. Software assurance by bounded exhaustive testing. *Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis*, p. 133–142, 2004.

[Smi18] Tom Smith. New research shows 63% of enterprises are adopting microservices architectures, 2018. URL: https://www.globenewswire.com/news-release/2018/09/20/1573625/0/en/New-Research-Shows-63-Percent-of-Enterprises-Are-Adopting-Microservices-Architectures-Yet-50-Percent-Are-Unaware-of-the-Impact-on-Revenue-Generating-Business-Processes.html.

[SVH98] Noemie Slaats, Bart Van Assche ir Albert Hoogewijs. *Shared memory synchronization. Proof in VDM: Case Studies*. J. C. Bicarregui, redaktorius. Springer London, London, 1998, p. 123–156. ISBN: 978-1-4471-1532-8. DOI: 10.1007/978-1-4471-1532-8_5. URL: https://doi.org/10.1007/978-1-4471-1532-8_5.

[War20] Eric Ward. Kafka controller doesn't failover during hardware failure. https://issues.apache.org/jira/browse/KAFKA-9957, 2020. Kafka problemų sistemoje paskelbtas sistemos sutrikimas.

[Whi00] J. A. Whittaker. What is software testing? and why is it so hard? *IEEE Software*, 17(1):70–79, 2000. DOI: 10.1109/52.819971.