„Redis Cluster“ podėlio sistemos tyrimas, taikant formalius metodus

Mantas Kontrimas, Karolis Petrauskas

Vilniaus universitetas, Matematikos ir informatikos fakultetas,
Informatikos institutas,
Didlaukio g. 47, LT-08303 Vilnius
mantas.kontrimas@mif.stud.vu.lt, karolis.petrauskas@mif.vu.lt

Santrauka. Šiame straipsnyje yra analizuojamas podėlio sistemos „Redis Cluster“ korektiškumas. Analizuojant sistemą buvo naudojami formalūs metodai – TLA$^+$ specifikavimo kalba buvo sudaryta sistemos formali specifikacija. Specifikacijos modelio tikrinimo metu buvo vertinama, ar yra užtikrinama sistemos savybė, kad už vieną maišos lizdą yra atsakingas tik vienas pagrindinis mazgas ir jo pavaldūs mazgai. Atlikus modelio tikrinimą buvo surastos situacijos, kada ši sistemos savybė nėra užtikrinama. Surastos klaidos buvo atkartotos realioje sistemoje ir šioms klaidoms buvo pateikti galimi sprendimo būdai.

Raktiniai žodžiai: Redis Cluster, Išskirstytos sistemos, Formalūs metodai, TLA$^+$.

1 Įvadas

Šiandien serveriai ir didesnės sistemos turi aptarnauti keletą tūkstančių klientų vienu metu, o vieno naudotojo puslapio užkrovimas gali pareikalauti keleto užklausų, kurios turi būti įvykdytos per sekundės dalis [1], [2]. Būtent todėl šiuolaikinėms sistemoms svarbu sumažinti atsako laiką į užklausas bei darbo krūvį, su kuriuo turi susidoroti serveriai [3]. Podėlio (angl. *cache*) operatyviojoje atmintyje naudojimas yra vienas iš efektyviausių būdų kaip galima tai pasiekti. Podėliui reikalingą struktūrą, susidedančią iš rakto-reikšmės poros, gali pasiūlyti atvirojo kodo produktai: „Redis“, „Memcached“, „Ehcache“ ir kiti [4], [5]. Taip pat, siekiant dar labiau pagerinti podėlio našumą bei patikimumą, podėlis gali būti paskirstytas arba replikuotas tarp mazgų (angl. *nodes*) [4], [6].

„Redis“ – tai viena iš populiariausių NoSQL duomenų bazių, operatyviojoje atmintyje laikančių raktas-reikšmė poras [7]. Dažniausiai ši duomenų bazė yra naudojama kaip podėlis, norint pagerinti sistemos našumą [8], [9].

DOI: https://doi.org/10.15388/LMITT.2021.4

Be to, „Redis“ palaiko ir blokinių kūrimą (angl. *clustering*), kuris leidžia padidinti podėlio talpą, duomenis paskirstant tarp keleto mazgų.

„Redis Cluster“ duomenis paskirsto skirtingiems mazgams, kurie yra vadinami pagrindiniais (angl. *master*) mazgais, o kiekvienas pagrindinis mazgas gali turėti keletą pavaldžių mazgų (angl. *slave*), kurie saugo pagrindinio mazgo duomenų kopijas. Duomenys „Redis Cluster“ sistemoje yra laikomi 16384 (2^{14}) maišos lizduose (angl. *hash slots*), kurių reikšmės yra apskaičiuojamos raktui, kuriuo norima padėti reikšmę pritaikant maišos funkciją. Kiekvienas mazgas yra atsakingas už tam tikrą maišos lizdų aibę. Blokinį sudarantys mazgai yra visi pilnai tarpusavyje sujungti komunikavimo kanalais, kuriais paskalų (angl. *gossip*) protokolo pagalba dalinasi blokinio būsena ir ilgainiui sutaria dėl kiekvieno mazgo būsenos [10].

Kadangi „Redis Cluster“ yra išskirstytas podėlis, tai šioje sistemoje gali egzistuoti išskirstytoms sistemos būdingos problemos [11]. Išskirstytose sistemose veiksmai vyksta lygiagrečiai, asinchroniškai, keičiasi mazgų būsenos, mazgai tarpusavyje komunikuoja žinutėmis, o ir sistemoje saugoma informacija yra išskirstyta tarp keleto mazgų. Būtent dėl to yra sunku užtikrinti, kad išskirstyta sistema nuolatos veiks korektiškai: bus užtikrintas duomenų neprieštaringumas (angl. *consistency*), sistema nepaklius į aklavietes (angl. *deadlock*) arba sistema bus funkciškai teisinga ir pasižymės kitomis jai būdingomis savybėmis, pateiktomis, pavyzdžiui, reikalavimų specifikacijoje [12], [13]. Norint tai užtikrinti, galima naudoti formalius metodus, kurie padeda ištirti sistemos korektiškumą ir jos savybes bei padeda surasti sudėtingas sistemos klaidas, kurių neaptinka kitos verifikavimo technikos [14].

Formalūs metodai – tai matematiniu pagrindu paremti metodai, dažniausiai naudojami aprašant sistemos savybes ir elgseną bei tiriant dviprasmybes, sistemos nepilnumą ir prieštaringumą. Sistemos kūrimo metu formalūs metodai gali būti naudojami specifikuojant sistemos reikalavimus, o sudarytos formalios specifikacijos gali padėti surasti sistemos klaidas, projektavimo trūkumus, prieš pradedant įgyvendinti pačią sistemą. Taikant formalius metodus jau sukurtai sistemai, formalūs metodai gali padėti pagrįsti sistemos įgyvendinimo korektiškumą [15]. Turint formalią specifikaciją, algoritmo ar sistemos korektiškumą galima patikrinti taikant formalios specifikacijos įrodymus, modelio tikrinimą, peržiūras, animacijas ir kitus metodus [16].

Taikant formalius metodus ir rašant formalias specifikacijas, yra naudojamos formalios specifikavimo kalbos, paremtos pasirinktu matematiniu

pagrindu, pavyzdžiui, algebra, logika, aibių teorija ir kitomis matematikos sritimis [17]. TLA$^+$ kalba – viena iš formalių specifikavimo kalbų, leidžiančių specifikuoti reaktyvias, išskirstytas ir asinchronines sistemas bei aptikti tokių sistemų klaidas [14], [18], [19]. Ši formali specifikavimo kalba paremta veiksmų laiko logika (angl. *temporal logic of actions*, sutrumpintai – TLA), kuri gali būti naudojama tiriant lygiagrečių ir išskirstytų sistemų veikimą bei savybes [20]. TLA$^+$ kalba nagrinėjamą sistemą aprašo kaip būsenų mašiną (išreikštą viena matematine formule), o modelio tikrintojas TLC tikrina visas galimas sistemos būsenas baigtiniame sistemos modelyje [18].

„Redis Cluster“ sistemos korektiškumas buvo vertinamas [4], tačiau nepaisant to, kad ši sistema yra plačiai naudojama, mokslinėje literatūroje ji tirta tik fragmentiškai. Atsižvelgiant į tai, šiame straipsnyje yra nagrinėjamas „Redis Cluster“ sistemos korektiškumas, taikant formalius metodus.

2 Formali specifikacija

Remiantis „Redis Cluster“ dokumentacija ir atvirai prieinamu „Redis Cluster“ sistemos (versija 5.0) programiniu kodu, TLA$^+$ kalba buvo sudaryta formali sistemos specifikacija, nagrinėjanti mazgų būsenos (informacijos apie kitus mazgus ir maišos lizdų atsakomybes) pasikeitimus. Galutiniame rezultate buvo gauta daugiau nei dviejų tūkstančių eilučių formali specifikacija artima programinio kodo abstrakcijos lygiui.

Formalioje specifikacijoje norint atitikti realios sistemos veikimą buvo modeliuojama žinučių komunikacija tarp blokinį sudarančių mazgų, t. y. vieno būsenų mašinos žingsnio metu būsena pasikeičia tik viename mazge ir informacija apie pasikeitimus pasiekia kitus mazgus, siunčiant žinutes paskalų protokolo pagalba.

Kiekvieno būsenų mašinos žingsnio metu gali įvykti vienas iš veiksmų:

1. Veiksmas susijęs su įvykiu ciklu (angl. *event loop*) – „Redis Cluster“ sistemoje visi veiksmai yra vykdomi vienoje gijoje, todėl visi veiksmai, susiję su žinučių apdorojimu, jų siuntimu, laiko įvykių apdorojimu, yra vykdomi įvykiu cikle.
2. Išoriniai įvykiai, lemiantys skirtingą žinučių apdorojimą ir sistemos veikimą:
   a. Veiksmas susijęs su laiko pasikeitimais – sistemoje daug veiksmų priklauso nuo laiko, todėl specifikacijoje buvo modeliuojama laiko tėkmė išreikšta tam tikru veiksmu, kuris arba įvyko arba ne.

b. Veiksmas susijęs su mazgo adreso pasikeitimu – blokinyje vienas iš mazgų gali pradėti bendrauti su kitais mazgais naudojant kitą adresą.
c. Veiksmas susijęs su blokinio administravimo komandų apdorojimu – tai veiksmai, leidžiantys keisti blokinio konfigūraciją: maišos lizdų pasiskirstymą, mazgų roles ar blokinio sudėtį.

Sudarius formalią specifikaciją, buvo apibrėžtas modelis (specifikacijos egzempliorius) skirtas tirti „Redis Cluster“ sistemos savybes. Modelis TLA$^+$ formalioje specifikavimo kalboje leidžia konfigūruoti formalią specifikaciją ir atlikti modelio tikrinimą esant skirtingoms specifikacijoje naudojamų konstantų reikšmėms, pradinei būsenai ir galimai būsenų mašinos būsenai. Sudarytas modelis yra naudojamas vykdant modelio tikrinimą ir generuojant sekančias būsenų mašinos būsenas. Apibrėžtą „Redis Cluster“ modelį sudaro trys pagrindiniai mazgai (mažiausia galima blokinio konfigūracija), kurie lygiomis dalimis yra pasidalinę šešis maišos lizdus. Siekiant sumažinti galimų būsenų skaičių, taip pat modelyje yra pridėtas ribojimas, kad komunikacijos kanaluose tarp mazgų negali būti daugiau nei trijų žinučių. Be to, norint pagreitinti modelio tikrinimą, pradinėje modelio būsenoje visi mazgai jau yra sudarę blokinį – nereikia vykdyti papildomos komunikacijos norint jį sukurti.

Sudarytos specifikacijos ir apibrėžto modelio tikrinimas buvo atliekamas nešiojamuoju kompiuteriu (4 branduoliai, 16GB RAM) bei TLA$^+$ Toolbox įrankiu (versija 1.7.1). Modelio tikrinimo metu buvo vertinama ar nėra pažeidžiamos sistemos savybės bei taip pat buvo tikrinamas tipų korektiškumas bei buvo tikrinama, ar sistema nepasiekia aklavietės. Naudojantis TLC įrankiu apibrėžto modelio tikrinimas buvo atliekamas tol, kol buvo surasta, kad yra pažeidžiama tikrinama savybė.

3 Tiriama „Redis Cluster“ savybė

Remiantis „Redis Cluster“ sistemos dokumentacija buvo išskirtos savybės, kurios turi būti užtikrintos, kad sistema veiktų korektiškai. Iš šių savybių buvo išsirinkta viena svarbiausių sistemos savybių – už vieną maišos lizdą gali būti atsakingas tik vienas pagrindinis mazgas ir jo pavaldūs mazgai. Šios savybės užtikrinimas bus vertinamas modelio tikrinimo metu.

Ši savybė yra svarbi, kadangi maišos lizduose yra saugomi klientų atsiųsti duomenys ir, susidarius situacijai, kai už vieną maišos lizdą yra atsakingi keletas mazgų, klientai gali prarasti anksčiau siųstus savo duomenis arba gali

nežinoti, į kurį mazgą reikia kreiptis dėl duomenų patalpinimo. Taip pat svarbu, kad visuose mazguose informacija, kuris mazgas yra atsakingas už kurį maišos lizdą, sutaptų. Tai reikalinga dėl to, kad „Redis Cluster" klientai gali kreiptis į bet kurį blokinį sudarantį mazgą bandydami gauti maišos lizde saugomas reikšmes. Apdorojant tokią užklausą mazgas, jeigu jis nėra atsakingas už tą maišos lizdą, klientui gali grąžinti *MOVED* klaidos pranešimą su nurodymu, į kurį mazgą reikia kreiptis dėl tame maišos lizde saugomos informacijos. Esant blogai informacijai apie mazgo maišos lizdus, „Redis Cluster" klientai gali nežinoti su kuriuo blokinio mazgu jiems iš tikrųjų reikia komunikuoti.

Maišos lizdų savybės užtikrinimas formalioje specifikacijoje buvo aprašytas invariantu. TLA$^+$ kalbos invariantą išreikštą predikatų logika galima pamatyti formulėje (1),

$$\forall s \in MAIŠOSLIZDAI, n, m \in MAZGAI.S(n) \land S(m) \land \neg M(n, s) \land \neg M(m, s) \\ \rightarrow \exists o \in MAZGAI.A(n, s, o) \land A(m, s, o) \quad (1)$$

kur $S(x)$ – mazgas x gali priimti skaitymo komandą, $M(x, y)$ – mazgo x požiūriu maišos lizdas y migruoja ir $A(x, y, z)$ – mazgo x požiūriu už maišos lizdą y yra atsakingas mazgas z. Šiuo invariantu yra tikrinama, kad bet kuriuose dviejuose mazguose n ir m, kurie gali priimti kliento komandas (yra pagrindiniai mazgai, jų požiūriu blokinys yra veikiantis ir dabar yra vykdomas žinučių skaitymo ir rašymo etapas, tačiau šiuo metu nėra apdorojama jokia žinutė), informacija apie mazgus ir jų maišos lizdų atsakomybes turi sutapti. Tikrinant šią savybę yra atsižvelgiama tik į pagrindinius mazgus dėl to, kad norint gauti tikslius duomenis „Redis Cluster" klientai turi kreiptis į pagrindinį mazgą. Taip pat, nėra tikrinami maišos lizdai, kurie dalyvauja maišos lizdų pertvarkyme (angl. *resharding*) – procese, kai maišos lizdai iš vieno mazgo yra perkeliami į kitą mazgą. Nors ir reali sistema gali aptarnauti tokius maišos lizdus naudojant komandą *asking* ir klaidos pranešimą *-ASK*, tačiau, norint supaprastinti invariantą, buvo atsižvelgta tik į atvejus, kada klientai gali naudoti komandas *get* ir *set* tam tikram maišos lizdui.

4 Tyrimo rezultatai

Naudojantis TLC modelio tikrintoju, buvo rasta atvejų, kai yra pažeidžiamas maišos lizdų invariantas (1). Rastos klaidos, jų priežastys ir pasiūlymai, kaip būtų galima jas ištaisyti pateikti žemiau.

1 klaida: Komanda *cluster setslot <maišos lizdas> node <mazgas>* **pažeidžia maišos lizdų savybę**

Maišos lizdų savybės pažeidimo priežastis – komandos *cluster setslot* subkomandą *node* galima iškviesti nurodant mazgą, kuris nedalyvauja maišos lizdų pertvarkyme. Sistema leidžia atlikti tokius veiksmus, nes viena iš subkomandos *node* paskirčių – rankiniu būdu pakeičiant maišos lizdų atsakomybes paspartinti informacijos apie įvykusį maišos lizdų pertvarkymą pasklidimą po „Redis Cluster" blokinį. Tačiau, kaip pavyko nustatyti, toks subkomandos veikimas pažeidžia maišos lizdų savybę.

Rasta klaida buvo atkartota realioje sistemoje. Buvo sukurtas „Redis Cluster" blokinys, kurį sudaro trys pagrindiniai mazgai: mazgas A, mazgas B ir mazgas C. Maišos lizdai mazgams buvo padalinti po lygias dalis. Turint tokią sistemos konfigūraciją ir naudojantis tekstine sąsaja (angl. *command line interface*) *redis-cli*, maišos lizdas 5061 buvo paskirtas mazgui B siunčiant subkomandą *node* mazgui A. Mazgui A apdorojus tokią komandą susidarė situacija, kad mazgo A požiūriu už 5061 maišos lizdą yra atsakingas mazgas B, o mazgo B požiūriu už šitą maišos lizdą yra atsakingas mazgas A. Esant tokiai situacijai per *redis-cli* nusiuntus komandą *get 5061* mazgui A yra gaunamas begalinis ciklas, kadangi mazgas A atsako, kad klientui dėl šio maišos lizdo reikia kreiptis į mazgą B, ir *redis-cli* pabandžius nusiųsti šitą *get* komandą mazgui B, jis atsako, kad klientui reikia kreiptis į mazgą A.

Norint ištaisyti šią klaidą būtų galima uždrausti kviesti *node* subkomandą, kai einamasis mazgas arba mazgas perduotas subkomandos parametruose nedalyvauja maišos lizdų pertvarkyme. Pašalintą *node* subkomandos funkcionalumą – paspartinti paskalų informacijos sklaidą apie pasikeitusius maišos lizdus, būtų galima užtikrinti šią atsakomybę perkeliant iš kliento į serverio pusę. Išsiuntus subkomandą *node* tiksliniam (angl. *target*) mazgui, kuris importuoja maišos lizdą, komandos apdorojimo metu tikslinis mazgas taip pat galėtų išsiųsti *ping* žinutes su pasikeitusiomis maišos lizdų atsakomybėmis visiems blokinio mazgams.

Perkėlus siūlomą pakeitimą į žinučių apdorojimo specifikaciją ir toliau vykdant modelio tikrinimą buvo pastebėta, kad subkomanda *node* vis tiek pažeidžia maišos lizdų invariantą. Kadangi informacija apie maišos lizdo migruojančią ir importuojančią būseną yra saugoma atitinkamai pirminiame (angl. *source*) ir tiksliniame mazge, tai subkomanda *node* leidžia nesilaikyti maišos lizdų pertvarkymo proceso veiksmų.

Šią klaidą būtų galima ištaisyti priverčiant klientus laikytis maišos lizdų pertvarkymo proceso veiksmų tvarkos. Tai būtų galima užtikrinti pradedant dalintis informacija apie maišos lizdo migruojančią būseną su visais blokinio mazgais – papildant paskalų informaciją. Tada tikslinis mazgas galėtų pakeisti maišos lizdo būseną į importuojančią tik tada, kai iš pirminio mazgo gavo informaciją, kad jis pakeitė norimo maišos lizdo būseną į migruojančią. Taip pat reiktų užtikrinti, kad subkomandą *node* būtų galima iškviesti tik tada, kai einamasis mazgas importuoja komandoje nurodytą maišos lizdą ir kai mazgas nurodytas subkomandoje *node* yra einamasis mazgas. Toks pakeitimas reikalingas norint užtikrinti, kad už maišos lizdų pertvarkymo proceso užbaigimą būtų atsakingas tik tas mazgas, kuris importuoja maišos lizdą. Kitu atveju, nesant šiam pakeitimui, būtų galima pasiekti situaciją, kai pirminis mazgas pakeičia maišos lizdo būseną į migruojančią ir iš karto užbaigią maišos lizdo pertvarkymo procesą gaudamas subkomandą *node*.

Įvykdžius visus siūlomus pakeitimus, maišos lizdo pertvarkymo procesas būtų vykdomas siunčiant komandas:

1. *Cluster setslot <maišos lizdas> migrating <tikslinis mazgas>* – ši komanda pakeistų maišos lizdo būseną į migruojančią ir ši informacija būtų pasidalinta su kitais mazgais paskalų protokolo pagalba.
2. *Cluster setslot <maišos lizdas> importing <pirminis mazgas>* – po pakeitimų šią komandą būtų galima iškviesti tik tada, kad buvo gauta paskalų informacija, kad maišos lizdas yra migruojančioje būsenoje.
3. *Migrate* – duomenys yra perkeliami iš pirminio mazgo į tikslinį mazgą.
4. *Cluster setslot <maišos lizdas> node <tikslinis mazgas>* – šią komandą būtų galima išsiųsti tik mazgui, kuris importuoją maišos lizdą. Taip būtų užbaigtas maišos lizdo pertvarkymo procesas, o pirminis mazgas išvalytų maišos lizdo migruojančią būseną tik tada, kai gautų paskalų informaciją, kad tikslinis mazgas (arba tikslinio mazgo pavaldus mazgas) tapo atsakingas už migruojantį maišos lizdą.

2 klaida: Komanda *cluster setslot* priima neteisingus parametrus

Tiriant 1 klaidos pasireiškimo aplinkybes, buvo rasta, kad yra pažeidžiamas maišos lizdų invariantas, kai komandos *cluster setslot* subkomandoms *importing* ir *migrating* yra perduodami blogi parametrai:

1. *Migrating* subkomanda leidžia parametruose perduoti tikslinį mazgą, kuris sutampa su einamuoju mazgu, kuriam yra siunčiama administravimo komanda.

2. *Importing* subkomanda leidžia parametruose perduoti pirminį mazgą, kuris sutampa su einamuoju mazgu.
3. Subkomandos *importing* parametruose galima perduoti maišos lizdą, už kurį nėra atsakingas komandoje nurodytas pirminis mazgas.

Visas šia klaidas pavyko atkartoti realioje sistemoje naudojantis *redis-cli*. Šias klaidas galima lengvai ištaisyti sistemoje pridedant papildomus patikrinimus, kad komandos yra kviečiamos perduodant teisingus parametrus.

3 klaida: Komandos *cluster addslots* ir *cluster delslots* pažeidžia maišos lizdų savybę

Modelio tikrinimo metu buvo rasta, kad komandų *cluster addslots* ir *cluster delslots* naudojimas pažeidžia maišos lizdų invariantą. Einamajame mazge apdorojus šias komandas, maišos lizdų atsakomybių pasikeitimas nepasiekia kitų mazgų.

Deja, bet šios klaidos nepavyko pilnai atkartoti realioje sistemoje – įvykus maišos lizdų pasikeitimams, dėl paskalų protokolo visada buvo sugrįžtama į pradinę būseną. Taip atsitiko dėl to, kad kitų mazgų požiūriu neįvyko jokie maišos lizdų pasikeitimai ir einamojo mazgo *configEpoch* reikšmė (skaičius nusakantis mazgo būsenos naujumą) buvo mažesnė ir nepakito – kitų mazgų *configEpoch* reikšmė buvo didesnė ir jų paskalų informacija laimėjo.

Norint ištaisyti šią klaidą būtų galima įvykus maišos lizdų pasikeitimams padidinti einamojo mazgo *configEpoch* reikšmę, kad kiti mazgai priimtų šią informaciją kaip naujausią. Kitas būdas taisyti šią klaidą: užtikrinti, kad šias komandas būtų galima apdoroti tik tada, kai sistema yra reikalingoje būsenoje, kurios metu apdorojus *addslots* ir *delslots* komandas nebūtų pažeista maišos lizdų savybė.

Ši sistemos klaida nėra kritinė kadangi „Redis Cluster“ dokumentacijoje nurodyta, kad šią komandą patartina siųsti tik tada, kai yra kuriamas naujas blokinys arba kai bandoma ištaisyti nenumatytas klaidas. Patys kūrėjai nurodo, kad šios komandos kvietimas nereikiamu metu gali palikti blokinį klaidingoje būsenoje, kaip ir buvo rasta modelio tikrinimo metu. Taip pat *cluster delslots* komanda nėra plačiai naudojama ir ji sistemoje buvo pridėta tik norint turėti pilnai užpildytą API sąsają.

5 Išvados

TLA$^+$ formalia specifikavimo kalba sudarius „Redis Cluster“ formalią specifikaciją ir atlikus modelio tikrinimą, buvo nustatyta, kad tam tikrose situacijose sistema gali veikti nekorektiškai – yra pažeidžiama sistemos maišos lizdų savybė (1). Pradinio modelio tikrinimo metu buvo identifikuotos sistemos klaidos, kurios buvo atkartotos realioje sistemoje naudojant tekstinę sąsają *redis-cli*. Šioms klaidoms buvo pateikti galimi sprendimo būdai, kaip jas būtų galima ištaisyti.

Detali sistemos specifikacija, artima programinio kodo abstrakcijos lygiui, leido surasti klaidas susijusias su blogais komandos parametrais. Rašant specifikaciją tik pagal sistemos dokumentaciją, šios klaidos galėjo būti nepastebėtos, kadangi nesiremiant programiniu kodu, būtų daroma prielaida, kad sistemoje egzistuoja korektiški komandos parametrų patikrinimai.

Sudaryta „Redis Cluster“ sistemos formali specifikacija leis toliau tirti ir kitas „Redis Cluster“ sistemos savybes, bei leis patikrinti, ar atlikus sistemos pakeitimus yra išlaikomas algoritmo korektiškumas ir nėra pažeidžiamos sistemos savybės.

Literatūra

[1] B. Veal and A. Foong. Performance scalability of a multi-core web server. In Proceedings of the 3rd ACM/IEEE Symposium on Architecture for Networking and Communications Systems, pp. 57–66, New York, NY, USA. ACM, 2007.

[2] V. Zakhary, D. Agrawal, and A. E. Abbadi. Caching at the web scale. Proceedings of the VLDB Endowment, 10:2002–2005, 2017.

[3] Y. Cheng, A. Gupta, and A. R. Butt. An in-memory object caching framework with adaptive load balancing. In Proceedings of the Tenth European Conference on Computer Systems, 4:1–4:16, New York, NY, USA. ACM, 2015.

[4] S. Chen, X. Tang, H. Wang, H. Zhao, and M. Guo. Towards scalable and reliable in-memory storage system: a case study with Redis. In 2016 IEEE Trustcom/BigDataSE/ISPA, pp. 1660–1667, 2016.

[5] B. Li, Y. Fu, and Z. Li. The research and improvement of distributee caching system Memcached. In 2017 4th International Conference on Information, Cybernetics and Computational Social Systems (ICCSS), pp. 460–463, 2017.

[6] A. Patil and R. Ingle. Leveraging information bus with in-memory caching for service oriented architecture. In 2013 Third International Conference on Advances in Computing and Communications, pp. 382–388, 2013.

[7] J. L. Carlson. Redis in action. Manning Publications Co., Greenwich, CT, USA, 2013. 5, 322 p.

[8] Z. Ji, I. Ganchev, M. O'Droma, and T. Ding. A distributed Redis framework for use in the UCWW. In 2014 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, pp. 241–244, 2014.

[9] D. Li, M. Dong, Y. Yuan, J. Chen, K. Ota, and Y. Tang. SEER-MCache: a prefetchable memory object caching system for IoT real-time data processing. IEEE Internet of Things Journal, 5:3648–3660, 2018.
[10] Redis. Redis cluster specification. https://redis.io/topics/cluster-spec, 2018. Kreiptasi 2019-05-19.
[11] D. Abadi. Consistency tradeoffs in modern distributed database system design: CAP is only part of the story. Computer, 45:37–42, 2012.
[12] F. A. Bianchi, A. Margara, and M. Pezzè. A survey of recent trends in testing concurrent software systems. IEEE Transactions on Software Engineering, 44:747–783, 2018.
[13] D. Malhotra. Deadlock prevention algorithm in Grid environment. MATEC Web of Conferences, 57:02013, 2016.
[14] C. Newcombe, T. Rath, F. Zhang, B. Munteanu, M. Brooker, and M. Deardeuff. How Amazon Web Services uses formal methods. Communications of the ACM, 58:66–73, 2015.
[15] J. M. Wing. A specifier's introduction to formal methods. Computer, 23:8–23, 1990.
[16] S. Liu. Validating formal specifications using testing-based specification animation. In Proceedings of the 4th FME Workshop on Formal Methods in Software Engineering, pp. 29–35, New York, NY, USA. ACM, 2016.
[17] N. A. Ali, A. A. Mirghani, and A. Y. Ibrahim. Alneelain: a formal specification language. In 2017 International Conference on Communication, Control, Computing and Electronics Engineering (ICCCCEE), pp. 1–9, 2017.
[18] L. Lamport, J. Matthews, M. Tuttle, and Y. Yu. Specifying and verifying systems with TLA+. In Proceedings of the 10th Workshop on ACM SIGOPS European Workshop, pp. 45–48, New York, NY, USA. ACM, 2002.
[19] O. Mosbahi. Combining formal methods for the development of reactive systems. ACM Transactions on Embedded Computing Systems, 12:16:1–16:29, 2013.
[20] L. Lamport. The temporal logic of actions. ACM Transactions on Programming Languages and Systems, 16:872–923, 1994.