VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ KATEDRA

Greito įvykių apdorojimo mechanizmų tyrimas .NET aplinkoje

Analysis of Fast Event Processing in .NET Environment

Magistro baigiamasis darbas

Atliko: Edmundas Malčius (parašas)

Darbo vadovas: Dr. Aistis Raudys (parašas)

Recenzentas: Karolis Uosis (parašas)

Vilnius – 2016

Santrauka

Darbe lyginami skirtingi metodai skirti išdalinti duomenis tarp atskirų gijų siekiant tai padaryti kiek įmanoma greitai ir be uždelsimo. Apžvelgiami septyni greičiausi metodai ir tikrinamas jų tinkamumas darbui su biržos duomenimis su skirtingu kiekiu apdorojančių gijų ir skirtingais duomenų apdorojimo lygiais. Apibrėžiama matavimų metodologija bei randami greičiausi metodai su skirtingo apdorojimo kiekio reikalaujančiais duomenimis.

Raktažodžiai: greitas įvykių apdorojimas, finansiniai duomenys, palyginimas, sparčiausias duomenų išdalinimo metodas.

Summary

In this Master thesis fast data distribution along different threads are analyzed. Seven fastest methods are overviewed and compatibility for market data is checked, also methods are benchmarked with different number of threads and different level of computations needed for data. Method for benchmarking methods is created. Methods are benchmarked and best one is found for each level of computations needed for new data.

Keywords: fast event processing, financial data, benchmark, fastest method for distributing data along threads.

Turinys

Įvadas

Technologinė pažanga leido kompiuterius naudoti daugelyje sričių: pradedant mobiliais telefonais, kurie gali atlikti visas įprastas asmeninio kompiuterio užduotis, baigiant dirbtiniu intelektu paremtomis sistemomis, kurios pačios geba priimti sprendimus pilotuojant lėktuvą, ar net valdydamos autonominį įrenginį (robotą, automobilį ar dulkių siurblį). Tokioms užduotims atlikti reikalingi ne tik dideli skaičiavimo resursai, tačiau ir kuo mažesnis uždelsimas, kitais žodžiais tariant, tokios sistemos turi veikti realiu laiku, nes net nedideli uždelsimai gali sukelti dideles, nepageidaujamas pasekmes.

Šiais laikais populiarėja procesoriai su daugiau nei vienu branduoliu. Norint išnaudoti tokį procesorių, reikia ir tam pritaikytos programinės įrangos. Programa turi būti sudaryta iš atskirų gijų. Programavimas naudojant gijas (ang. *Multithreading*) buvo naudojamas ir tada kai procesoriai buvo sudaryti iš vieno branduolio, tačiau tai nebuvo naudojama paspartinti darbui, o tiesiog norint atlikti keletą skirtingų užduočių vienu metu, pavyzdžiui: grafinėje vartotojo sąsajoje paspaudus mygtuką vartotojas toliau galėtų naudotis programa, kad ji „neužlūžtų“, kol bus atliktas norimas veiksmas. Vieno branduolio procesoriuje tik atrodo, kad skirtingos gijos veikia lygiagrečiai, o iš tikrųjų operacinė sistema išskiria trumpą laiko tarpą tai gijai, paskui perduoda procesorių kitai gijai ir taip kartojasi, kas vartotojui sudaro įspūdį, kad viskas vyksta lygiagrečiai [Rei]. Daugiabranduolinio procesoriaus atveju skirtingos gijos iš tikrųjų gali veikti lygiagrečiai, tačiau programa tam turi būti paruošta - išlygiagretinta, tai yra procesoriaus resursų reikalaujantis darbas turi būti paskirstytas į atskiras gijoms, kurių skaičius lygus ar didesnis nei branduolių skaičius, nes nepadarius to, bus apkraunamas tik vienas branduolys [Rei]. Išlygiagretinant užduotis daugiausiai problemų iškyla įgyvendinant bendravimą tarp atskirų gijų. Įmanomi du būdai, kaip įgyvendinti tokią komunikaciją: vienas jų naudojant bendrą atmintį, kitas, kai atskiros gijos bendrauja per žinutes. Bendros atminties atveju reikalingi užraktai, kurie vienu metu leidžia tik vienai gijai patekti į tam tikrą atminties zoną ir atlikti joje pakeitimus, taip priverčiant kitas gijas, kurios taip pat nori ten patekti, laukti kol vykdančioji gija pabaigs darbą [Rei]. Iš čia kyla problema, kad laukiančios gijos galėtų dirbti kitus darbus, o ne tuščiai laukti, nes tai padidina instrukcijų skaičių procesoriui norimam rezultatui pasiekti. Naudojantis žinučių komunikaciją sunaudojama daugiau atminties, nes tokios komunikacijos įgyvendinimui reikalingos papildomos duomenų struktūros.

Pastaruoju metu vis plačiau taikomi įvykių srauto apdorojimo ir kompleksinio įvykių apdorojimo metodai (ang. Event stream processing ir complex event processing). Įvykio srauto apdorojimo metodai taikomi duomenims, kurie gaunami iš vieno pastovaus šaltinio [BGA], o

kompleksinio įvykių apdorojimo metodai taikomi, kai duomenys gaunami iš skirtingų šaltinių [Bas]. Praktikoje tokie mechanizmai taikomi kaip tarpininkai tarp įvykių šaltinio ir programos dalies, kuri atlieka tam tikrus sprendimus, t.y. toks mechanizmas pateikia jau apdorotą ir naudingą informaciją [Obe]. Kaip pavyzdį būtų galima paimti autonominį automobilį, kuris renka informaciją iš įvairių daviklių: vaizdo kameros, kad „suprastų“ kelių ženklus, lazerinių ir ultragarsinių jutiklių, kurie leidžia automobiliui „matyti“ aplinką, GPS imtuvo informaciją apie padėti ir daug kitų, kurią apdoroja, ir jei yra reikalas kažkaip pakeisti automobilio kursą ar greitį, perduoda šią informaciją sistemos daliai, atsakingai už automobilio mechaninių dalių valdymą.

Tokių metodų įgyvendinimas taip pat priklauso ir nuo procesoriaus architektūros bei platformos, o kartais ir nuo operacinės sistemos, ant kurios šie metodai yra įgyvendinti. Nors tokių sprendimų yra įgyvendintas ne vienas, paprastai tai lieka kaip verslo paslaptis, o sukonkretinus tokio sprendimo dalykinę sritį alternatyvų arba išvis nėra, arba jos komercinės ir architektūriniai sprendimai nėra prieinami, nes tokio mechanizmo sukūrimas reikalauja ne tik teorinių žinių, ar žmogaus resursų, bet ir nemažai bandymų, testavimų, kurie užima nemažai laiko.

Viena iš nedaugelio sistemų, kurios architektūra ir programinis kodas yra prieinamas visiems norintiems, yra „LMAX Exchange“ elektroninės prekybos brokeris. Sistema sukurta Java programavimo kalba ir veikia ant standartinės Java virtualios mašinos. Kūrėjai savo sistemos branduolį pavadino LMAX architecture. Sistema geba atlikti 6 milijonus transakcijų per sekundę su 1 milisekundės uždelsimu. Panašaus tipo sistemos viešai prieinamos architektūros ar programinio kodo kitoms platformoms atrasti labai sunku, todėl šiuos tyrinėjimus atliksiu magistriniame darbe.

1.1 Darbo tikslas

Internete netrūksta bendro pobūdžio informacijos apie algoritminę prekybą, netrūksta ir pavienių algoritmų. Gaila, tačiau sudėtingų prekybos algoritmų, naudojančių daug skirtingų prognozavimo strategijų, ir architektūrinių sprendimų, kurie leidžia apdoroti kiek įmanoma daugiau įvykių vienu metu, viešai pasiekti negalima, o taip pat neretai jie būna pritaikyti veikimui su skirtinga techninė ir programine įranga. Taip pat nėra informacijos apie greitų įvykių apdorojimo metodų palyginimus, strategijas kaip tai atlikti.

Mano darbo tikslas – parengti greitų įvykių apdorojimo geriausiojo metodo išrinkimo eksperimentų strategiją.

Darbui iškelti tokie uždaviniai:

1) Ištirti skirtingas greitam įvykių apdorojimui taikomas architektūras, joms taikomus algoritmus ir jų sukūrimo būdus.

2) Sukurti automatizuotos prekybos agento prototipą, panaudojant skirtingus algoritmus greitam įvykių apdorojimui realizuoti.

3) Palyginti skirtingų algoritmų rezultatus naudojant bendrojo pobūdžio skaičiavimus ir be jų.

4) Palyginti skirtingų algoritmų plečiamumo galimybes didinant branduolių skaičių.

1.2 Laukiami rezultatai

Sukūrus taikomąja programą, bei išanalizavus jos gautus rezultatus tikimasi įvertinti greito įvykių apdorojimo architektūrų tinkamumą algoritminei prekybai, palyginti skirtingų algoritmų, skirtų realizuoti greitą ivykių apdorojimą, tinkamumą naudoti su bendrojo pobūdžio skaičiavimais panaudojant procesorių su keliais branduoliais, įvertinti plečiamumą bei pateikti metodiką, greitam įvykių apdorojimo architektūrų taikymui bei vertinimui

2. Greitasis įvykių apdorojimas

Šiais laikais įmonių sistemos yra sudėtingesnės negu bet kada ankščiau. Skirtingi procesai vyksta visame pasaulyje ir įvykiai sklinda po skirtingas IT sistemas. Šios sistemos išaugo iš mažų, nepriklausomų, programų, kurios gebėjo apdoroti tik tam tikrus įmonės aspektus, į dideles, plataus masto informacines sistemas, kurios apjungia skirtingas informacinių technologijų taikymo sritis.

Kai kalbama apie informacinių technologijų sistemas, atsiranda daug įvairių terminų ir sutrumpinimų. Patys naujausi iš jų yra SOA (Service-Oriented Architecture) and EDA (Event-Driven Architecture). Tai dvi architektūros, kurios leidžia apjungti skirtingas programas, integruoti procesus ar išskirstyti per skirtingas sistemas.

Šios technologijos yra plačiai paplitusios tarp didelių įmonių, kur sugeneruojamas didelis kiekis įvairių įvykių, kurie pasiekia skirtingus sistemos sluoksnius, tačiau atrandamas jų panaudojimas ir kitose srityse, pavyzdžiui mano nagrinėjamoje srityje apie algoritminę prekybą. Įvykius apdoroja kitos programos ar servisai, kurių rezultatas yra naujai sugeneruoti nauji įvykiai. Toks reiškinys kartais vadinamas įvykių debesiu. Įvykių debesys kartais sunkiai perprantamas, nes paprasti įvykiai yra nesunkiai atsekami, bet sudėtingi įvykiai, kuriuos sudaro keletas nesusijusių įvykių, atsekti pasidaro sunku. Tokių įvykių apdorojimui įvestas naujas trumpinys – CEP (Complex Event Processing), kompleksinių įvykių apdorojimas arba greitasis įvykių apdorojimas. Tai metodai skirti prižiūrėti, apdoroti ir reaguoti į kompleksinius įvykius realiu laiku. Kai atsiranda kažkokia problema ar galimybė, reikia, kad būtų informuota nedelsiant, kad būtų galima laiku imtis teisingo sprendimo, nes kitaip tai liks tik istoriniais duomenimis, kurie įvardina galimą problemą, arba galimybes, kurios bus praleistos. Kompleksinis įvykių apdorojimas įgalina į tai reaguoti realiu laiku, taip pat geriau išnaudoja ir apdoroja jau esamus įvykius sistemose.

Dar viena sritis kurioje sėkmingai taikomas kompleksinių įvykių apdorojimas greitam įvykių apdorojimui, yra automatizuota algoritminė prekyba [SLS]. Šiais laikais automatizuotos prekybos agentai užimą nemažą dalį tarp visų rinkoje prekiaujančių dalyvių, ir tas procentas nuolat auga. Tokios veiklos automatizavimas turi nemažai pliusų prieš prekiaujantį žmogų: pro kompiuterį „nepraslys“ nei vienas kainos pakitimas, taip pat kompiuteris rinką gali stebėti ištisai, be pertraukimo, ilgą laiką. Vienas kompiuteris gali sužiūrėti daug daugiau prekybos instrumentų ir jų pokyčių, nei žmogus, sistemoje taip pat nebelieka žmogiškojo faktoriaus, tai yra sistema neturi polinkio rizikuoti, taip apsisaugodama nuo didelių nuostolių, nes viskas vyksta pagal nustatytas taisykles, sistema taip pat geba reaguoti į rinką daug greičiau, nei žmogus. Būtent spartus įvykių apdorojimas algoritminėje prekyboje tinka kaip dalykinė sritis ir pabrėžia temos aktualumą, nes laisvai prieinamų prekybinių sistemų architektūrų, kurios naudotų greitą įvykių apdorojimą, nėra.

2.1 Panaudojimas

Greitas įvykių apdorojimas gali būti naudojamas įvairiems tikslams patenkinti. Šiame skyriuje trumpai apžvelgsiu kelias panaudojimo sritis:

- Įvykiais paremtos architektūros (Event-Driven Architecures)
- Įmonių sistemų integracija (Enterprise Application Integration)
- Verslo procesų valdymas (Business Process Management)
- Verslo veiklų priežiūra (Business Activity Monitoring)

2.1.1 Įvykiais paremtos architektūros

Įvykiais paremta architektūra, tai tokia programinės įrangos infrastruktūra, kurios dalys yra silpnai apjungtos. Principas, kad tai yra didelė programų sistema sudaryta iš daug mažų komponentų, kur kiekvienas jų turi savo funkcijas. Tokio tipo yra ir mano nagrinėjama sritis, algoritminė prekyba. Komunikacija tarp atskirų komponentų atliekama siunčiant įvykius. Įvykis gali būti paprasčiausias pranešimas pasakantis, kad kažkokia užduotis jau yra atlikta. Šioje architektūros taip pat labai svarbu ir įvykių apdorojimas ir nukreipimas. Kompleksinis įvykių apdorojimas gali puikiai papildyti tokios architektūros galimybes, nes geba aptikti sudėtingas situacijas realiu laiku. Tokių įvykių svarba taip pat paminėta ir [WebM]: *"Kadangi įmonės naudoja įvykiais paremtas architektūras, verslo veiklų priežiūra ir procesų valdymo iniciatyvos diktuoja reikalavimus kompleksinių šablonų atpažinimui iš skirtingų įvykių, kurie įvyksta organizacijos įvykių matricoje. Kadangi tokių įvykių skaičius yra milžiniškas, tokius įvykius atpažinti ir suprasti realiu laiku, kas leistų atlikti kritinius verslo sprendimus, yra sudėtinga.."*

1 pav. Įvykiais paremtų sistemų architektūrų modelis

2.1.2 Įmonių sistemų integracija – EAI

Įmonių sistemų integracijos terminas apibrėžiamas skirtingai. Globalus apibrėžimas iš [Seab] yra toks: „*EAI – tai verslo skaičiavimų terminas naudojamas planams, metodams ar įrankiams skirtiems modernizuoti, įtvirtinti ir koordinuoti kompiuterines programas įmonėse.*“

Šiuolaikinės įmonės naudoja skirtingų tipų sistemų, pavyzdžiui: CRM (customer realtionship management) , SCM (Supply Chain Management) and BI (Business intelligence) programas. Tokios sistemos talpina daug informacijos, taip pat į jas investuota didelės sumos pinigų. EAI – tai būdai apjungti tokias sistemas, taip sukuriant naujas programas. Naudojantis EAI metodais skirtingų sistemų duomenys gali būti saugomi nuosekliai.

2.1.3 Verslo procesų valdymas – BPM

Verslo procesų valdymas, tai koncepcija apjungti vadybą ir informacines technologijas. BPM apima verslo procesus, organizacijas, žmones ir sistemas. Į BPM taip pat įeina trys veiklos:

- Procesų kūrimas
- Procesų paleidimas
- Procesų priežiūra

Vadyba pateikia žinias verslo procesų kūrimui, informacinės technologija jų paleidimą (panaudojimą), o procesų priežiūrą apima BAM (Business Activity Management) apie kurį pakalbėsiu kitame skyriuje.

BPM apibrėžimas [Seaa]: *„BPM – tai sistematinis požiūris į verslo procesų gerinimą. BPM veiklos siekia verslo procesus padaryti efektyvesniais, naudingesniais ir labiau prisitaikyti prie pastoviai kintančios aplinkos.“*

2.1.4 Verslo veiklų priežiūra - BAM

Verslo veiklų priežiūra yra pagalbinis įrankis, leidžiantis stebėti verslo darbą ir padeda identifikuoti silpnas vietas. BAM sudarytas iš trijų žingsnių:

- Duomenų rinkimas
- Duomenų apdorojimas
- Rezultatų pateikimas

Kompleksinis įvykių apdorojimas, naudojimas verslo veiklų priežiūroje, yra labai naudingas dalykas, nes geba aptikti sudėtingas situacijas, vykstančias didelėje verslo aplinkoje, kas leidžia BAM įrankiams pateikti detalesnę informaciją. Taip pat [WebM] pateikiama, kad naujausias BAM lygis turi kompleksinių šablonų atpažinimą.

2.2 Įvykių įvairovė

Didelėse sistemose įvykių gali būti daug skirtingų tipų, iš skirtingų lygių, pavyzdžiui pradedant žemo lygio tinklo lygio įvykiais ir baigiant aukšto lygio verslo įvykiais. Analogiškai algoritminėje prekyboje tokie įvykiai galėtų būti:

- Kainos pokytis
- Indikatorių pranešimai apie galima prekiavimo signalą
- Naujienos, įtakojančios rinką
- Sudėtingų genetinių algoritmų pranešimai apie prekybos signalą

2.3 Įvykių gavimas

Įvykiai yra pagrindinė kompleksinių įvykių apdorojimo (CEP) variklio dalis, todėl labai svarbu, kad visus šiuos įvykius sistema gautų, ir nei vieno nepraleistų, kad būtų priimtas teisingas sprendimas, arba jis nebūtų praleistas. Verslo sistemose tokie įvykiai gali būti gauti įvairiais būdais, algoritminėje prekyboje įvykių šaltiniai ir gali būti kelių rūšių.

2.4 Kompleksinių įvykių šablonai

Šablonai naudojami tam, kad būtų galima aptikti tam tikrą konkrečią situaciją, algoritminės prekybos atveju, tam tikrą prekybos signalą. Pats šablonas, tai keletas įvykių, pagal kuriuos galima atlikti tam tikrą konkretų sprendimą. Šablonų aptikimas yra viena iš kompleksinių įvykių apdorojimo mechanizmo sudedamųjų dalių.

3. Greitų įvykių apdorojimo metodų panaudojimas

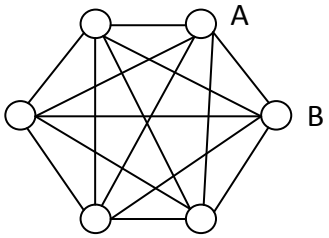
Šiame skyriuje bus apžvelgti kompleksinio įvykių apdorojimo mechanizmai gali būti integruoti tarp skirtingų įvykių šaltinių.

3.1 Įvykių šaltinių topologijos

Apžvelgsiu keletą skirtingų sujungimo būdų, į kuriuos gali būti įterptas CEP mechanizmas.

3.1.1 Taškas į tašką

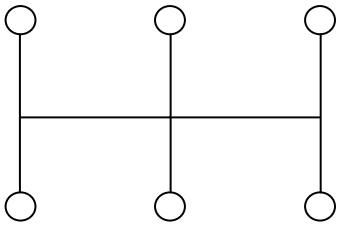
Taškas į tašką, tai tokia tinklo topologija, kai kiekvienas mazgas vienas su kitu yra sujungtas jungtimis. Jei visi mazgai yra sujungti vienas su kitu, tai turint n mazgų, jungčių skaičius bus n*(n-1) ir toks jungimas kartais vadinamas „spageti“ sujungimu. Tokiame tinkle esantys mazgai bendrauja tiesiogiai vienas su kitu. Mazgas A verčia savo duomenis mazgui B suprantamu protokolu, o mazgas B atvirkščiai. Grafinis pavyzdys:



2 pav. Taškas į tašką topologija

3.1.2 Magistralė

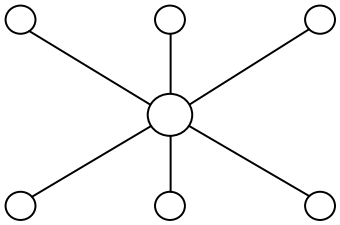
Magistralė, tai tokia tinklo topologija, kai mazgai naudoja bendrą magistralę bendravimui vienas su kitu. Bendraujantys mazgai žinutes verčia į bendrą magistralės protokolą. Kitas privalumas, kad tokiai topologijai, sudarytai iš n mazgų, reikia n jungčių. Pavyzdys:



3 pav. Magistralės topologija

3.1.3 Žvaigždė

Žvaigždė, tai tokia tinklo topologija, kai egzistuoja centrinis mazgas, prie kurio prisijungę visi kiti magai. Visas mazgų bendravimas vyksta per centrinį mazgą, konvertuojant žinutes į protokolą, kurį supranta centrinis mazgas. Pavyzdys:



4 pav. Žvaigždės topologija

3.2 Kompleksinių įvykių apdorojimo mechanizmų įterpimas

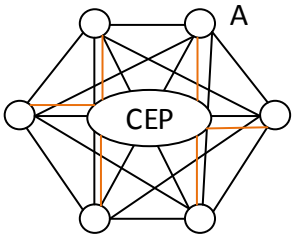
Kompleksinių įvykių apdorojimo mechanizmui, labai svarbu, kad visi įvykiai pasiektų jį. Svarbiausi punktai, kuriuos turi įvykdyti CEP variklis yra:

- Gauti visus reikiamus įvykius, iš visų šaltinių
- Galimybė siųsti įvykius į visus skirtingus šaltinius (algoritminėje prekyboje nebūtinai į visus)
- Suprasti įvykius iš visų skirtingų šaltinių

3.2.1 Įterpimas į „taškas į tašką“ topologiją

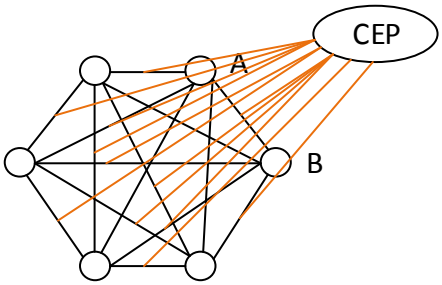
Kaip jau buvo apžvelgta ankščiau, visi mazgai turi jungtį su kiekvienu kitu mazgu. Iš čia kyla bėda, kad siunčiant įvykį iš vieno mazgo į kitą, kiti mazgai to įvykio negauna, todėl kompleksinių įvykių apdorojimo atveju, mechanizmas turi gauti kiekvieno siunčiamo įvykio kopiją, o tai reiškia kad kiekvienas mazgas dar papildomai turi būti sujungtas su šiuo CEP varikliu.

Antra iškylanti problema, kad tokioje topologijoje visi mazgai naudojasi savo protokolu, ir verčia jį į siunčiamajam mazgui suprantamą, todėl nėra bendro – universalaus komunikacijos protokolo, todėl papildomai, kiekvienas mazgas informaciją dar turėtų paversti į CEP mechanizmui suprantamą protokolą, tai yra turėtų atskirą mechanizmą tam pavertimui, todėl auga tokio tinklo sudėtingumas ir painumas.



5 pav. Įterpimas į "taško į taško" topologiją

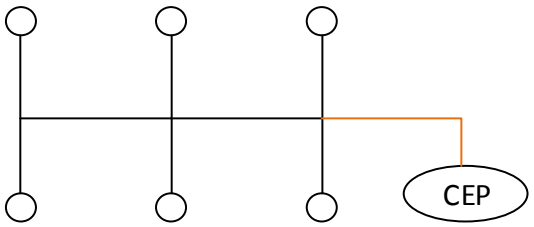
Taip pat galimas sprendimas praplėsti jau esamas jungtis ir kiekvieną sujungimą sujungti su CEP varikliu, tokiu atveju CEP variklis turėtų daugiau įeinančių linijų, tačiau nereiktų kurti atskiro mechanizmo transliavimui į CEP suprantamą formatą, užtektų papildyti esamą keitimo mechanizmą, nes tokį keitiklį būtų galima įgyvendinti kiekviename mazge papildomai prie to, kuris jau verčia į kitam mazgui suprantamą protokolą.



6 pav. Įterpimas į "taško į taško" topologiją

3.2.2 Įterpimas į magistralės tipo topologiją

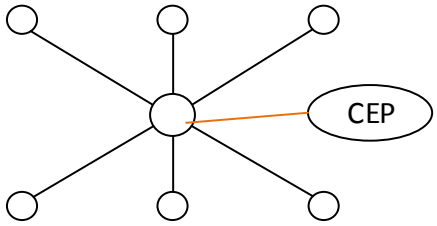
Tokioje topologijoje kompleksinių įvykių apdorojimo mechanizmą įterpti nėra sunku: CEP prijungiamas prie magistralės ir vienintelis keliamas reikalavimas, kad CEP suprastų magistralės bendravimo protokolą, kad galėtų siųsti ir gauti įvykius.



7 pav. Įterpimas į magistralės topologiją

3.2.3 Įterpimas į žvaigždės tipo topologiją

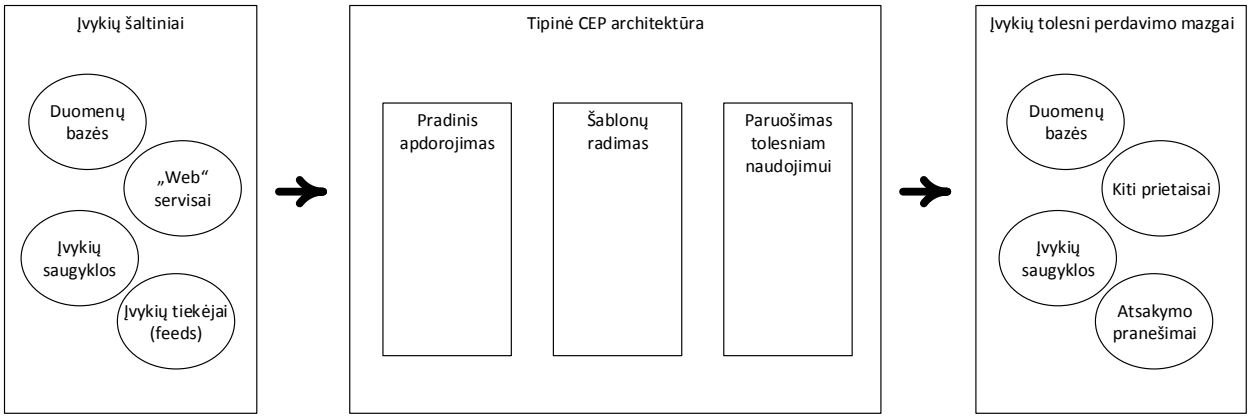
Literatūroje teigiama, kad kompleksinių įvykių apdorojimo mechanizmo integravimas į tokią topologiją yra paprasčiausias, nes užtenka variklį prijungti prie centrinio mazgo, o patį mazgą papildyti taip, kad jis persiųstų visą informaciją į CEP variklį. Vienintelis trūkumas, panašus į kitų topologijų sukeliamas problemas, kad reikalingas atskiras protokolas, kuriuo būtų įgalinta komunikacija tarp CEP ir centrinio mazgo.



8 pav. Įterpimas į žvaigždės topologiją

4. Modelis

Literatūroje nėra konkrečios greitųjų įvykių apdorojimo kūrimo architektūros, standartų ar reikalavimų, kaip tai turi būti įgyvendinta. Kitais žodžiais tariant, terminas greitųjų įvykių apdorojimas (CEP) naudojamas įvardinti metodams, kurie įvykius apdoroja ir norimą rezultatą pasiekia su minimaliu uždelsimu arba be jo, taikant įvairius algoritmus ir duomenų struktūras, kurios leidžia efektyviai išnaudoti techninę įrangą. Šiame skyriuje apžvelgsiu bendro pobūdžio greitų įvykių apdorojimo modelį ir keletą taikomų metodų jam įgyvendinti.



9 pav. Greitų įvykių apdorojimo modelis

Kaip jau minėta ankščiau, įvykių šaltiniai tam tikra forma pateikia duomenis, kompleksinių įvykių apdorojimo mechanizmui suprantamu protokolu. Įvykiai gali būti tiekiami pačio šaltinio, arba jie gali būti tiekiami su užklausa, pvz: internetiniai servisai (webservices).

Pačiame kompleksinių įvykių apdorojimo mechanizmo viduje, daugeliu atvejų, bet ne visada, būna dalis, atsakinga už pradinį įvykių pradinį apdorojimą, kurios tikslas prieš įvykiams patenkant į pagrindinį apdorojimą, paversti pranešimus į mechanizmui suprantamą formatą, ar papildyti kažkokia informacija. Paruošta informacija yra atfiltruojama pagal norimus reikalavimus, o aptikus šabloninį kompleksinį įvykį šis nusiunčiamas apdorojimui, kurio tikslas paruošti rezultatą ar naują įvykį tolesniam naudojimui ar persiuntimui.

4.1.1 Šablonų filtravimas

Nors nėra bendro būdo, kaip kurti greito įvykių apdorojimo mechanizmus, šablonų atpažinimo metodai daug kur naudojami panašūs. Šablonų atpažinimas yra svarbiausia CEP variklio dalis, leidžianti priimti sprendimą ar identifikuoti tam tikrą situaciją. Atpažinimas dažniausiai apima kelis metodus, tai filtravimas, koreliacijų paieška, grupavimas ar kaupimas iš gaunamo įvykių srauto.

4.1.1.1 Koreliacijos

Pirmas žingsnis yra sugrupuoti gautus įvykius, taip sudarant „langą“. Tuomet įvykiai yra koreliuojami pritaikant techniką, vadinamą „langų politika“:

- Laikini langai, arba tiesiog laiko laikai, tai įvykių, turinčių būseną, koreliacija, priklausanti nuo įvykio nutikimo laiko ar kitų aplinkybių. Priklausomai nuo laiko yra sukuriamas įvykių srautas ir tikrinama prieš tai buvusių įvykių būsena su dabartiniu įvykiu, kad būtų galima

nustatyti šabloną. Pavyzdys: finansinio instrumento kaina pasikeitė 10 procentinių punktų nuo to laiko, kai ji buvo nupirkta.

- Edvine koreliacija arba dimensijomis paremti langai yra panašūs į laikinus langus, skirtumas tas, dėmesys kreipiamas įvykių kiekiui, o ne laikui. Kitaip dar ši technika vadinama skaičiavimų langu, nes nuo įvykių skaičiaus priklauso langas. Pavyzdys: iš eilės įvyko trys vienodi įvykiai.
- Tiesioginiai filtrai. Pavyzdys: ieškomos konkrečios valiutų poros, pavyzdžiui EUR/USD.

Laikini ir dimensiniai langai sudaromi ir sunaikinami pagal tam tikras sąlygas. Įvykiai, dar kitaip vadinami „kortežais“, lango viduje iškeldinami pagal tam tikrus iškeldinimo principus. Jei visi įvykiai yra pašalinti iš lango dar prieš sukuriant naują langą, tai toks langas vadinamas, „nuosekliuoju langu“. Kartais iš lango būna pašalinami tik seniausi kortežai, ir taip lango sąlyga tampa tenkinama, tokie langai vadinami „slankiojančiais langais“.

1	2	3	4	5	6	7	8	9	10	11	12	...

10 pav. Įvykių srautas

Šiame sraute, kiekvienas langelis yra tam tikras įvykis. Pasirinkus 3 elementų nuoseklų langą, srauto įvykiai bus sugrupuoti taip:

1	2	3	4	5	6	7	8	9	10	11	12	...

11 pav. Nuoseklus langas

Tokio pat dydžio slankiojantis langas atrodys taip:

1	2	3										
	2	3	4									
		3	4	5								
			4	5	6							
				5	6	7						
					6	7	8					
						7	8	9				
							8	9	10			
								9	10	11		
									...	...	...	

12 pav. Slenkantis langas

Skirtingos spalvos žymi naujus langus. Atėjus naujam įvykiui, senasis yra pašalinimas, kad būtų galima išlaikyti tą patį lango dydį, šiuo atveju 3.

4.1.1.2 Bendrų įvykių skaičiavimas

Kai įvykiai sukoreliuojami, atliekamas tolesnis skaičiavimas ir filtravimas. Kaip jau minėta ankščiau, naudojantis langų metodais įvykiai sugrupuojami, ir nepriklausomai nuo lango tipo, ieškoma bendrų įvykių. Pavyzdys:

Turime įvykių srautą:

1	2	3	4	5	6	7	8	9	10	11	12	...

13 pav. Įvykių srautas

Tarkime, kad skaičius langelyje žymį to įvykio svorį, ir ieškomas langas, kurio svorių vidurkis yra 3, kai langą sudaro 3 elementai.

Nuoseklaus lango atveju vidurkiai suskaičiuojami taip:

Langas			Vidurkis
1	2	3	2
4	5	6	5
7	8	9	8

14 pav. Vidurkio ieškojimas nuosekliame lange

Jei lango tipas slenkantis, tuomet skaičiuojama taip:

Langai										Vidurkis
1	2	3								2
	2	3	4							3
		3	4	5						4
			4	5	6					5
				5	6	7				6
					6	7	8			7
						7	8	9		8
							8	9	10	9

	9	10	11		10
		...	...	...	....

15 pav. Vidurkio ieškojimas slenkančiame lange

Dauguma CEP platformų jau turi paprastų funkcijų skaičiavimų realizacijas, kaip kad sum(), avg(), min(), max(). Kai kurios platformos naudoja ir statistinius metodus standartiniui nuokrypiui ar medianai rasti ir pan. Taip pat CEP karkasuose dažniausiai būna integruota tam tikra kalba, leidžianti aprašyti taisykles, pagal kurias bus ieškoma įvykių.

Kiekvienas naujai gautas įvykis apdorojimas ir įsimenama jo būsena, kad nebūtų atlikti pakartotiniai skaičiavimai.

4.1.2 Atmintis

Dauguma greitų įvykių apdorojimo mechanizmų, siekiant užtikrinti greitą darbą, naudoja tik operatyviąją atmintį, ir į kietąjį diską kreipiasi tik tada, kai to reikia ir neįmanoma be to apsieiti.

RAM atmintis pagrinde naudojama laikyti langus ir įvykius, tačiau taip pat naudojama ir dalintis informacija tarp srautų, taip pat būsenos saugojimui ar saugoti duomenis iš išorinių šaltinių, kad būtų galima greitai apdoroti. Šie duomenys saugomi atskirai nuo langų ir įvykių srautų, tačiau reikalui esant gali būti naudojami. Reikia pažymėti, kad išjungus programą, pasišalina ir duomenys iš operatyviosios atminties, jei jie niekur neišsaugomi. Norint neprarasti duomenų turi būti įgyvendinti papildomi mechanizmai, leidžiantys duomenis saugoti ir kietajame diske. Tokių mechanizmų naudojimas stipriai padidina CEP metodų prieinamumą (availability).

Darbo spartinimui naudojamos tokios technikos:

- Least Frequently Used (LFU)
- Least Recently Used (LRU)
- First in First out (FIFO) – kai maksimaliai užpildoma spartinančioji atmintinė.

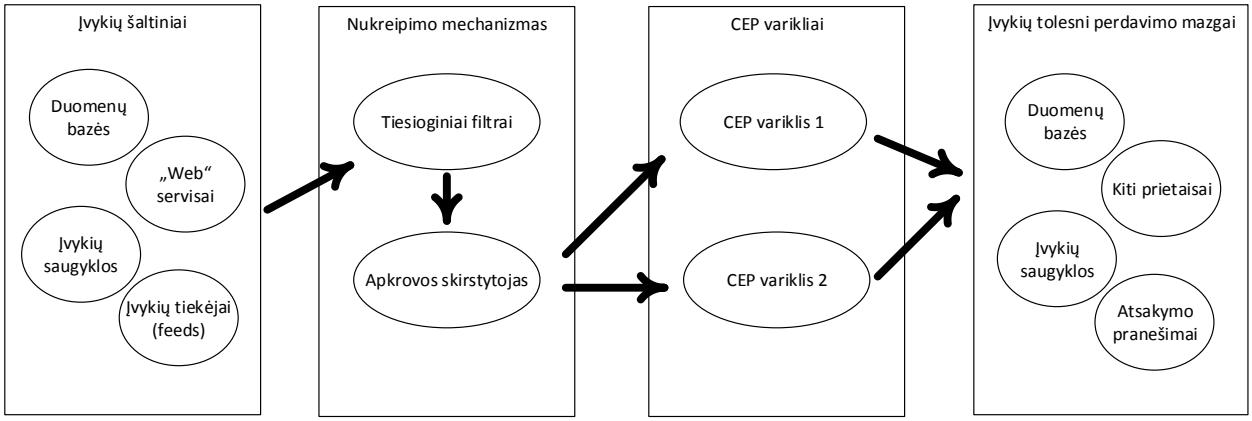
4.1.3 Apimtys ir delsa

Bendrą kompleksinių įvykių greito apdorojimo metodų spartą taip pat riboja ir duomenų kiekis. CEP metodai turi apdoroti didelius kiekius duomenų su labai mažu uždelsimu, kai kalba pasisuka apie itin didelius duomenų kiekius (terabaitai ir daugiau), tuomet žvilgsniai krypsta į didelių duomenų platformas. Dauguma CEP metodų savyje turi ir įrankius, kurie leidžia analizuoti veikimo spartą ir identifikuoti vietas, kuriose kyla didžiausias uždelsimas.

4.1.4 Plečiamumas

Kadangi pagrindinės kompleksinio įvykių apdorojimo metodų charakteristikos yra duomenų kiekis ir uždelsimas, didėjant duomenų kiekių CEP metodai turi vis tiek veikti didele sparta. Tokie modeliai turi būti sukurti taip, kad būtų plečiami, ir didėjant procesorių kiekiui, gebėtų išnaudoti jų teikiamus privalumus išlygiagretinant darbą tarp atskirų procesoriaus branduolių. Nors lygiagretimas ir išsprendžia daug klausimų susijusių su sparta, tačiau pasiekus procesorių galimybių ribas, reiks ieškoti naujų sprendimų.

Kai platforma neleidžia išlygiagretinimo, galima naudoti atskirus CEP variklius, kurių veikimo principas yra sekantis: duomenys iš pradžių praeina tiesioginius filtrus, tada apkrovos skirstytojas nukreipia srautą į atitinkamą CEP elementą.



16 pav. Keletos CEP variklių išnaudojimas

Vėliau rezultatai apjungiami ir siunčiami toliau.

4.1.5 Prieinamumas

Greitų įvykių apdorojimų metodų tikslas - greitai atlikti sprendimus, todėl tokie metodai turi būti prieinami kiek įmanoma didesnį laiko tarpą. Taip pat šie metodai turi savo būseną, todėl svarbu, kad įvykiai nepasimestų.

Būsenos buvimui užtikrinti naudojama:

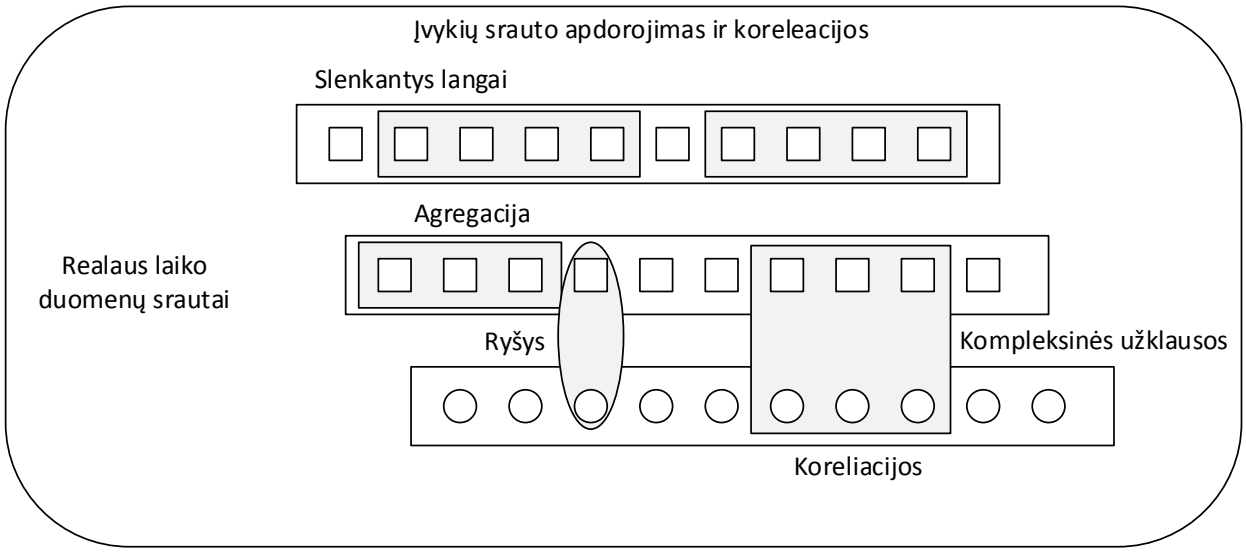
- Akyvus/Aktyvus – replikuojama būsena ir funkcionalumas iš vieno mazgo į kitą. Privalumas - trumpas trykių laikas ir gera sparta.
- Aktyvus/Pasyvus – replikuojama tik būsena iš vieno mazgo į kitą. Privalumas – mažesnis apkrovimas, nes tik vienas mazgas atlieka visą reikiamą funkcionalumą.
- Tarpiniai taškai – išsaugoma būsena tole
- sniam naudojimui. Privalumas – lengvas įgyvendinimas.

5. Esami karkasai

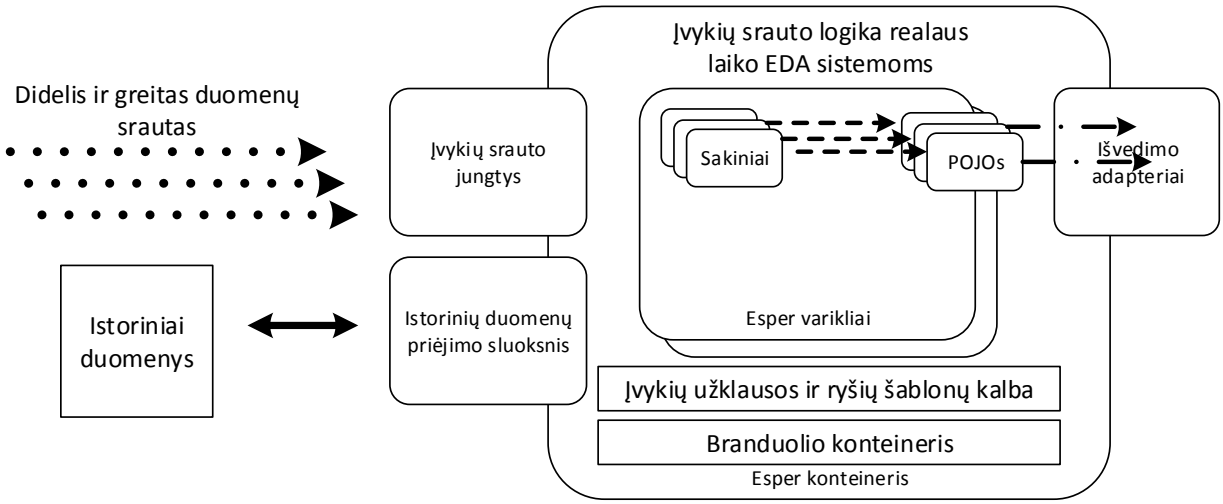
5.1 Atvirojo kodo

5.1.1 Esper

Esper – vienas geriausiai žinomų atvirojo kodo greitų įvykių apdorojimo variklis, kuris gali būti taikomas tiek su Java kalba, tiek ir su C# (NEsper). Projektas prasidėjo 2004 metais, spalio mėnesį, o pirmoji „alfa“ versija pasirodė 2006 sausio mėnesį. Šablonams sudaryti ir manipuliuoti duomenims naudojama jų pačių sukurta kalba - EQL (Event query language). Architektūros pagrindas – slenkantys langai.



17 pav. Esper architektūra



18 pav. Esper karkasas

Esper taip pat turi sluoksnį, kuris leidžia prisijungti prie visų populiarių duomenų bazių, kad būtų galima gauti istorinius duomenis. Gali būti lengvai integruojamas su visais populiariais serveriais, JAVA ir .NET technologijomis.

5.1.2 Kiti varikliai

Taip pat egzistuoja keletas kitų laisvai prieinamų CEP variklių, kurie buvo kurti eksperimentiniais tikslais ir toliau nebeplėtojami, taip pat neturintys ir architektūros aprašymo:

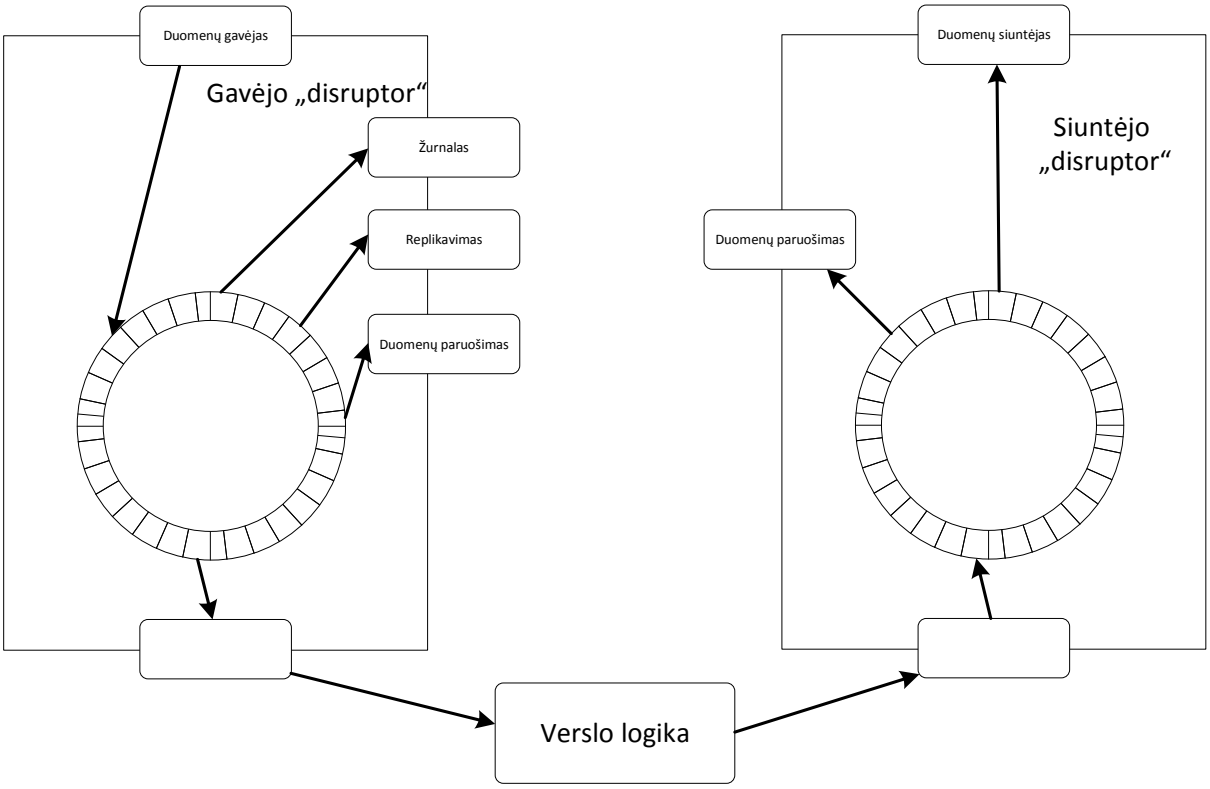
- Siddhi CEP
- TriCEPs

5.1.3 Atvirojo kodo projektai, naudojantys greitą įvykių apdorojimą

Čia apžvelgsiu vieną programinės įrangos projektą, kurio pagrindinės architektūros kodas yra viešai prieinamas, ir nors tai nėra specializuotas karkasas, programinė įranga naudoja specialiai jai sukurtą greitų įvykių apdorojimo mechanizmą.

5.1.3.1 LMAX biržos brokerio programinė įranga

LMAX-Exchange, tai Anglijos biržos brokeris, teikiantis elektroninės prekybos paslaugas. Transakcijoms apdoroti buvo kuriama įranga, kurios kūrėjai orientavosi į galimybę apdoroti didelį kiekį užklausų su minimaliu uždelsimu, nes esami sprendimai netiko. Rezultate programinė įranga, veikianti serveryje su 3GHz 4 branduolių procesoriumi ir 32GB operatyviosios atminties, geba atlikti 6 milijonus transakcijų per sekundę. Kūrėjai nesiorientavo į jau esamus modelius, ir nesistengė jų pagerinti, bet bandymų keliu bandė sukurti savo, geresnį už jau esamus. Savo architektūrą jie pavadino LMAX-architecture. Ši architektūra skiriasi nuo tradicinio CEP modelio, kurį apžvelgiau ankstesniuose skyriuose.

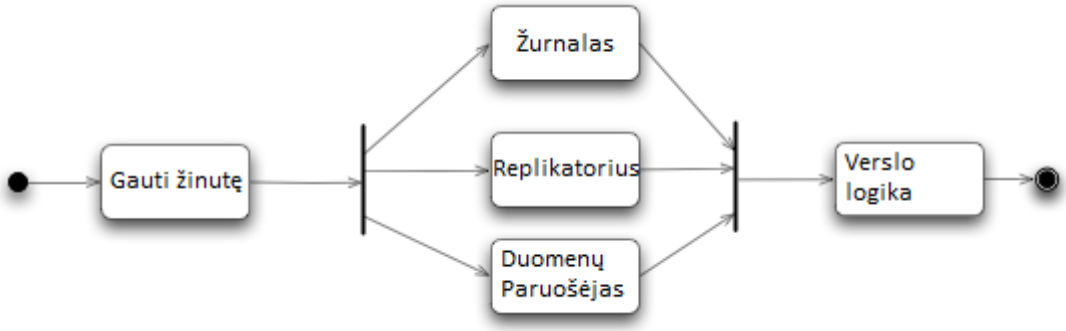


19 pav. LMAX architektūra

Sistema sudaryta iš specialių elementų, kuriuos kūrėjai pavadino „disruptor“ šablonu. Šie elementai atlieka apdorojimą gavus tam tikrus įvykius ir juos išsiunčiant tolesniam naudojimui, o verslo logikai pateikiama tik pati svarbiausia informacija, kuri atliekama nuosekliai. Skirtumų nuo tradicinio CEP modelio priežastis yra ta, kad tai yra produktas, orientuotas į konkrečią sritį (užklausų aptarnavimą) su aiškiomis įeitimis ir išeitimis, ir skirtas ne atrasti kažkokius įvykius sraute, o tiesiog juos greitai apdoroti su minimaliu uždelsimu.

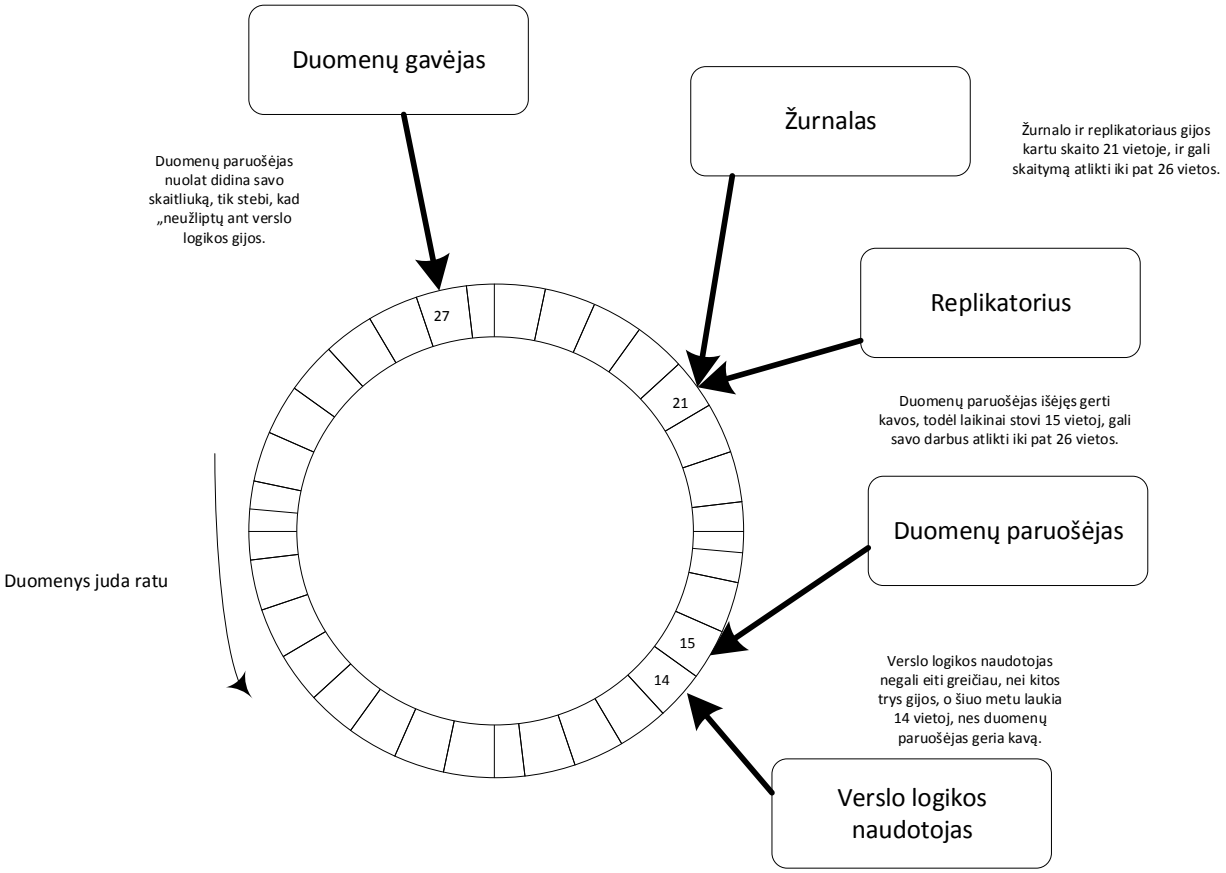
5.1.3.1.1 „Disruptor“ šablonas

Nors verslo logika ir veikia, kaip vienas neišlygiagretintas procesas, gavus naują įvykį būtina atlikti keletą kitų veiksmų, prieš patenkant į verslo logikos apdorojimą. Pirmas dalykas, gautoji žinutė serverį pasiekia kaip kažkoks užkoduotas pranešimas, todėl jį reikia paversti į verslo logikai suprantamą formą, taip pat visi pranešimai yra žymimi, todėl gautasis pranešimas turi būti patalpinamas į įvykių žurnalą tinkama forma ilgalaikiam saugojimui, taip pat, kadangi verslo logiką sudaro tam tikras klasteris procesorių, gautos žinutės turi būti replikuojamos per visus šio apdorojimo mazgus. Panašūs procesai vyksta ir išėjimo „disruptor“ atveju - išeinantys įvykiai taip pat turi būti paversti atitinkamos formos pranešimu.



20 pav. Įeities "disruptor" veiklų UML diagrama

Įeities „disruptor“ atveju turime tris darbus, kurie gali būti atlikti nepriklausoma tvarka, tačiau visi jie turi būti pabaigti iki patenkant į verslo logiką. Kūrėjai, siekiant išvengti tokios būsenos, kai procesas laukia kažkokių užrakintų resursų, vietoj to, kad atliktų naudingą darbą, sukūrė „disruptor“ šabloną. Tai žiedinės formos buferis, kuriame kiekvienas gamintojas ir naudotojas (producer and consumer) turi savo eilės numerį, kad galėtų parodyti, kokioje buferio vietoje jis dabar dirba. Kiekvienas kūrėjas, įrašinėja savo eilės numerį, tačiau gali skaityti ir kitų kūrėjų ar naudotojų eilės numerius, o tai leidžia žinoti, ar vieta, į kurią norimą įrašyti duomenis, prieinama be jokių užraktų (locks). Panašiai ir naudotojai gali matyti, ar jau galima pradėti apdorojimą, pavyzdžiui jei tą resursą prieš tai buvo laikinai pasiėmęs kitas naudotojas.



21 pav. Disruptor šablonas

5.2 Uždaro kodo sprendimai

Egzistuoja uždaro kodo, komerciniai sprendimai:

- Microsoft SteamInsight
- IBM® InfoSphere® Streams
- Oracle event processing
- Aleri RAP

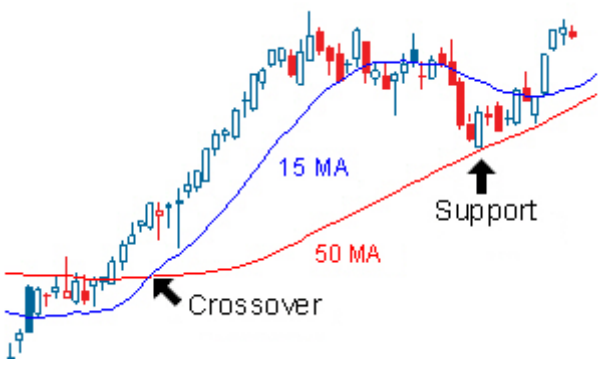
6. Eksperimentai

Šiame skyriuje aprašysiu atliktus praktinius eksperimentus, programinius sprendimus, eksperimentų vertinimo metodiką.

6.1 Eksperimentų tikslas ir dalykinė sritis

Dalykinė sritis, kuriai bus atliekami eksperimentai – algoritminė prekyba, ir su akcijų biržomis susijusi informacija. Algoritminė prekyba - tai programinės įrangos pagalba automatizuoti agentai, atliekantys prekybos veiksmus akcijų biržose, atsižvelgiant į tam tikras užprogramuotas taisykles, rinkos sąlygas duomenų analizes ir kitus veiksnius, veikiančius rinkas. Bendruoju požiūriu, biržoje prekiaujantis žmogus ir biržoje prekiaujantis programinis agentas vienas nuo kito nesiskiria. Abu vertina rinką, pasitelkdami tas pačias priemones:

- Techninė analizė – būdas numatyti ateities kainų judėjimą analizuojant praeities duomenis
    - Techniniai indikatoriai – matematiniai skaičiavimai, kurie remdamiesi praeities duomenimis gali nustatyti esamą ar tolesnę rinkos tendenciją. Rezultatas dažniausiai pateikiamas kaip grafikas ar kreivė. Pats paprasčiausias indikatorius – slenkantis vidurkis.



22 pav. Du slenkantys vidurkiai. Paimta iš www.investopedia.com

    - Neuroniniai tinklai – tai sparčiai populiarėjantis metodas, rinkos prognozavimui įkvėpti biologinių neuroninių tinklų (žmogaus smegenų). Tokie tinklai dažniausiai išmokomi atpažinti tam tikrą šabloną, kurio atpažinimas leidžia imtis tam tikrų veiksmų rinkoje.
    - Atgalinis testavimas – sisteminga prekyba pagal istorinius duomenis, norint patikrinti ar strategija gera, senesniais laikais tai buvo atliekama grynai rankomis.

- Fundamentalioji analizė – finansinės analizės būdai ir metodai, kurių tikslas, nustatyti finansinių vienetų (vertybinių popierių) vertę. Tokia analizė remiasi ne biržos duomenimis, tačiau fundamentaliaisiais duomenimis. Tokios analizės tikslas nustatyti vertę ir prognozuoti jos kitimą.
    - Rinkos analizė – vertinama šalies ekonominė padėtis ir ekonominė politika taip pat politinė situacija ir socialinė padėtis.
    - Šakos analizė – šakos struktūrų ir charakteristikų taip pat gamybinių veiksmų ir pagrindinių ekonominių ir gamybinių veiksmų analizė bei šakos perspektyvų tyrimai. Taip nustatomas konkrečios šakos konkurencingumas nustatant toje šakoje veikiančių įmonių perspektyvas.
    - Įmonės analizė – nuodugnus įmonės ūkinės ir finansinės veiklos rezultatų tyrimas. Ši analizė grindžiama nuostata, kad akcijos vertę sąlygoja ją išleidusios veiklos efektyvumas, tačiau akcijų vertė priklauso ne tik nuo įmonės pelningumo, bet ir nuo rizikos laipsnio.

Fundamentalioji analizė taip pat gali būti skaitmenizuota ir panaudota automatizuotoje algoritminėje prekyboje, kaip ir pavieniai metodai gali būti automatizuojami tačiau galutinis sprendimas atliekamas žmogaus. Apjungus mano paminėtus kainos prognozavimo metodus gaunama prekybos sistema.

23 pav. Prekiautojo langas su indikatoriais EUR/RUB valiutų porai, 5 minučių grafikui

Įvykus kainos ar kitos informacijos pokyčiui, tam tikros prekybos sistemos dalys, kurios tiekia susijusią informaciją, reikalingą sprendimui priimti, turi būti perskaičiuotos iš naujo. Prekiaujančio žmogaus atveju sekundės uždelsimas didelės įtakos nepadarytų, nes strategijos dažniausiai orientuotos į ilgalaikę prekybą, tačiau automatizuotos prekybos sistemos atveju, ypatingai tokios, kurios orientuotos į momentinius kainos pokyčius, kiekviena uždelsta milisekundė gali reikšti prarastą pelną ar net nuostolius, todėl labai svarbu, kad atskiros sistemos dalys, naujausią kainą gautų kuo greičiau, kad galėtų perskaičiuoti prognozę su minimaliu uždelsimu. Savo darbe autorius simuliuos skirtingus metodus su realiais duomenimis ir dalinai aprašytu prekybos agentu, kurie leidžia išdalinti kainos pokyčius per atskiras sistemos dalis minimaliu uždelsimu.

6.2 Eksperimentų modelis

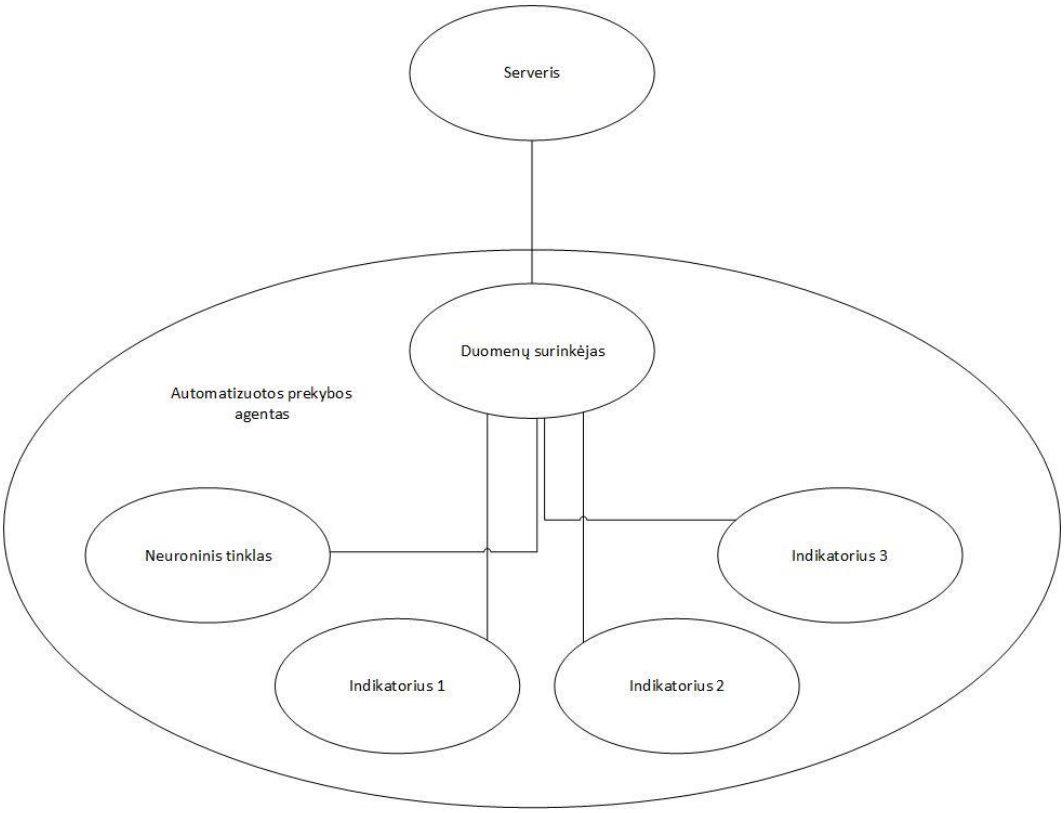
Bendruoju automatizuoto prekybos agento atveju yra dalis atsakinga už duomenų paėmimą iš serverio, vėliau tie duomenys padalinami visoms agento dalims, kad būtų galima perskaičiuoti skaičiavimus, būtinus sprendimui priimti. Duomenys, kurie ateina iš serverio yra:

- Kaina
- Laikas ir data
- Sandorių kiekis

Eksperimentų metu duomenys generuojami atsitiktiniu būdu, nes tyrinėjama dalis yra būtent duomenų išdalinimas skirtingoms programos dalims, ir norint patikrinti, kaip skirtingi metodai elgiasi su skirtinga apdorojimo apkrova, bus atliekami slenkančio vidurkio skaičiavimai skirtingo dydžio periodams ir lango dydžiams.

6.2.1 Apkrova

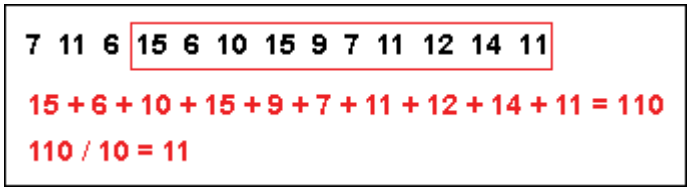
Skirtingi metodai naudoja skirtingus algoritmus, kurie reikalauja skirtingo kiekio procesoriaus ir atminties resursų, siekiant kuo greičiau sureaguoti į kainos pakitimą, todėl eksperimentų metu dirbtinai susimuliuosiu atminties ir procesoriaus apkrovimą didinant duomenų kiekį ir slenkančio vidurkio skaičiuojamąjį periodą, kad būtų galima patikrinti, kaip skirtingi metodai veikia, prie skirtingai apkrauto procesoriaus.

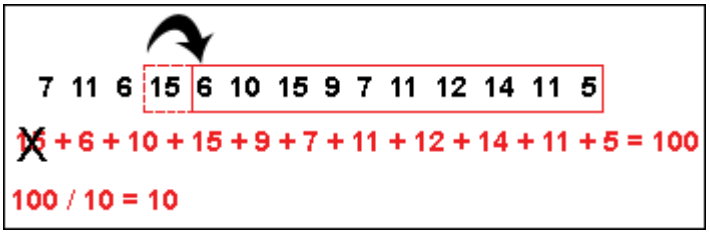


24 pav. Duomenų gavimo ir paskirtymo modelis

6.2.2 Slenkantis vidurkis

Vieni labiausiai naudojamų įrankių finansinių duomenų techninei analizei atlikti yra slenkantys vidurkiai. Algoritminės prekybos agentą galima sukurti, apjungiant kelis skirtingais periodais skaičiuojančius slenkančius vidurkius. Slenkantis vidurkis – tai pasirinkto kiekio (periodo) istorinių duomenų suvidurkinimas. Kaip ir signalų apdorojime taip ir biržos duomenys, srautas dažniausiai būna triukšmingas, bet paimus šios ir dviejų ankstesnių dienų kainas, kainos šokinėjimai bus suvidurkinti (Pav 4.) Naudojant daugiau duomenų, galime gauti suvidurkintą kainą, kuri mums parodytų kainos kitimo tendenciją, pašalinant smulkius svyravimus.





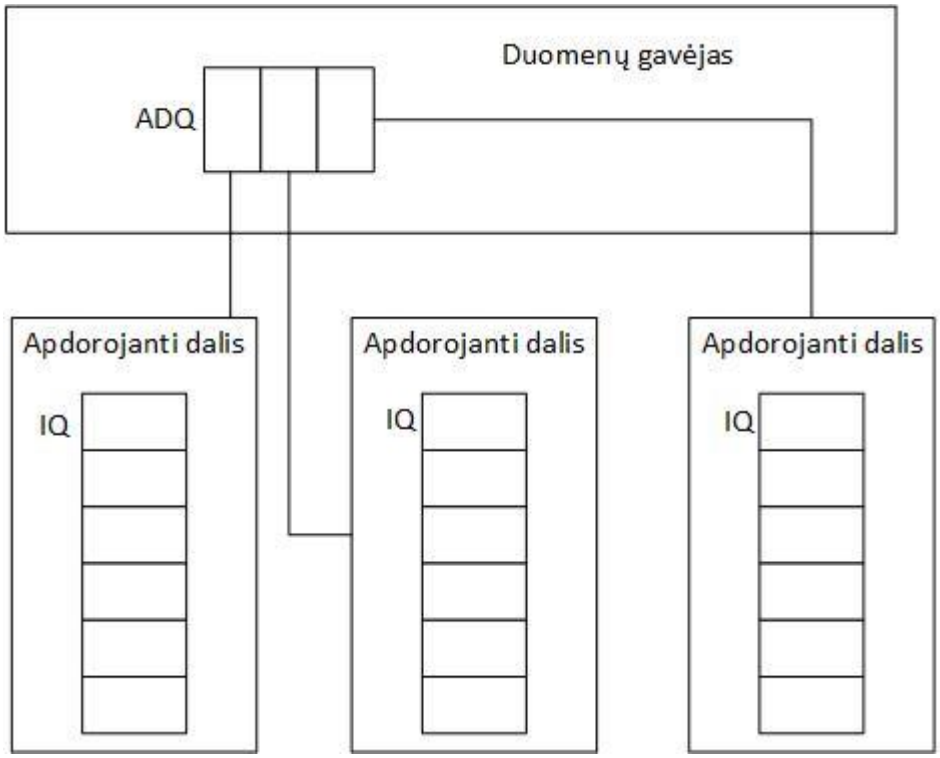
25 pav. Paprastojo vidurkio skaičiavimas

Paprastasis slenkantis vidurkis (SMA - Simple moving average) - plačiausiai žinomas ir labiausiai naudojamas. Šį vidurkį apskaičiuoti paprasta, tiesiog suskaičiuojamas paprastas vidurkis pagal nurodytą periodo kainų vidurkį. Šį vidurkį ir panaudosiu savo darbe

$$SMA = \frac{p_M + p_{M-1} + \cdots + p_{M-(n-1)}}{n}$$

6.2.3 Atskira „concurrent“ eilė kiekvienai apdorojančiai agento daliai

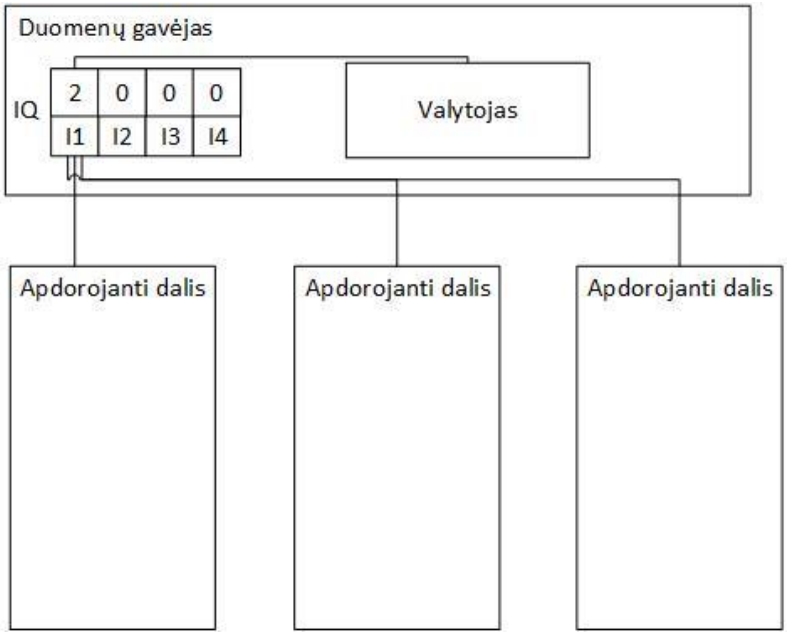
Šiame modelyje kiekviena apdorojanti dalis turi savo „concurrent“ eilę IQ, į kurią įrašomi naujausi kainos pokyčiai. Už duomenų įrašymą į šias eiles yra atsakingas duomenų gavėjas kuris turi sąrašą ADQ nuorodų, į visas apdorojančias dalis. „Concurrent“ eilė jau yra realizuota .NET karkase ir jos veikimas pagrįstas veikimu be užrakto naudojant „SpinWait“ algoritmą, kurio metu procesas neužmigdomas, o tiesiog atlieka labai mažos apimties ciklą ir taip nuolatos gaudamas procesoriaus, gali patikrinti ar resursas jau atsilaisvino. .NET platformoje ši eilės realizacija atsirado su 4 karkaso versija. Šį metodą į darbą įtraukiau dėl to, kad tai vienas dažniausiai internete nurodomų sprendimo būdų siekiant greitai ir efektyviai išdalinti duomenis atskirom gijom.



26 pav. Atskiros "concurrent" eilės

6.2.4 Bendra „concurrent" eilė

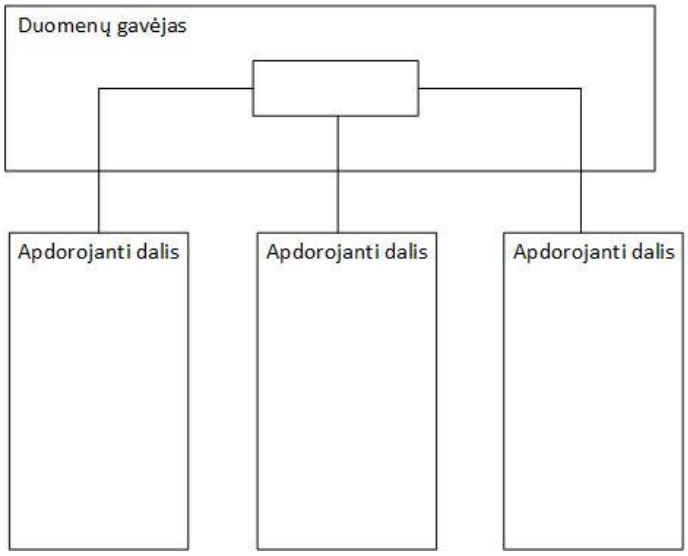
Šiame modelyje naudojama viena bendra „concurrent" eilė, kurioje saugomi nauji kainų pasikeitimai ir kiekviena apdorojanti dalis kreipiasi į eilę, norint gauti kainų pasikeitimus. Papildomai kiekviena apdorojanti dalis turi užsižymėti, kad pasiėmė elementą ir prisiminti paskutinį elementą, kad nepaimtų jo dar kartą. Papildomai taip pat atsiranda ir dar vienas elementas - „valytojas" kuris tikrina, ar visos apdorojančios dalys jau pasiėmė elementą ir jei taip, tuomet jį pašalina iš eilės. Įtraukiau į darbą šį metodą dėl to, kad jis atmintyje užima mažiau vietos nei atskiromis eilėmis pagrįstas metodas.



27 pav. Bendra "concurrent" eilė

6.2.5 .NET4 lygiagretinimo įrankiai.

Šiame modelyje išnaudojama .NET 4 platforma ir su ja atsiradę įrankiai padedantys lengvai įgyvendinti užduotis, reikalaujančias lygiagretaus vykdymo, apie juos plačiau 5.5.1 skyriuje. Duomenų gavėjas, gavęs naujus kainos rinkos pasikeitimus, naudojantis Parallel.ForEach komanda paleis visas apdorojančias dalis veikti lygiagrečiai, gijoms baigus apdorojimą jos baigs ir savo vykdymą ir gavus naujus pasikeitimus vėl bus paleistos iš naujo. Toks sprendimas pasirinktas todėl, nes .NET platformos kūrėjai rekomenduoja naudoti šią biblioteką darbui su lygiagrečiomis užduotimis ir teigia, kad toks gijų paleidimas ilgai neužtrunka.



28 pav. .NET lygiagretinimo įrankiai

6.2.6 „Disruptor“ šablonas

Tai be užraktų veikiantis šablonas, kuris buvo sukurtas biržos brokerio tarnybinei stočiai, kad būtų galima aptarnauti kuo daugiau klientu su minimaliu uždelsimu. Šablonas atvirojo kodo, todėl prieinamas visiems norintiems ir nors originaliai kūrėjai kūrė šį šabloną tik JAVA platformai, tačiau trečiųjų šalių dėka šis šablonas perrašytas visomis populiariomis programavimo kalbomis, tarp jų ir C#.

„Disruptor“ šablonas tai žiedinės formos buferis, kuriame kiekvienas gamintojas ir naudotojas (duomenų gavėjas ir apdorojančios dalys mano atveju) turi savo eilės numerius, kad galėtų parodyti, kokioje buferio vietoje jis dabar dirba. Kiekvienas kūrėjas (mano atveju tik vienas), įrašinėja savo eilės numerį, tačiau gali skaityti ir kitų kūrėjų ar naudotojų eilės numerius, o tai leidžia žinoti, ar vieta, į kurią norimą įrašyti duomenis, prieinama be jokių užraktų (locks). Panašiai ir naudotojai gali matyti, ar jau galima pradėti apdorojimą, pavyzdžiui jei tą resursą prieš tai buvo laikinai pasiėmęs kitas naudotojas.

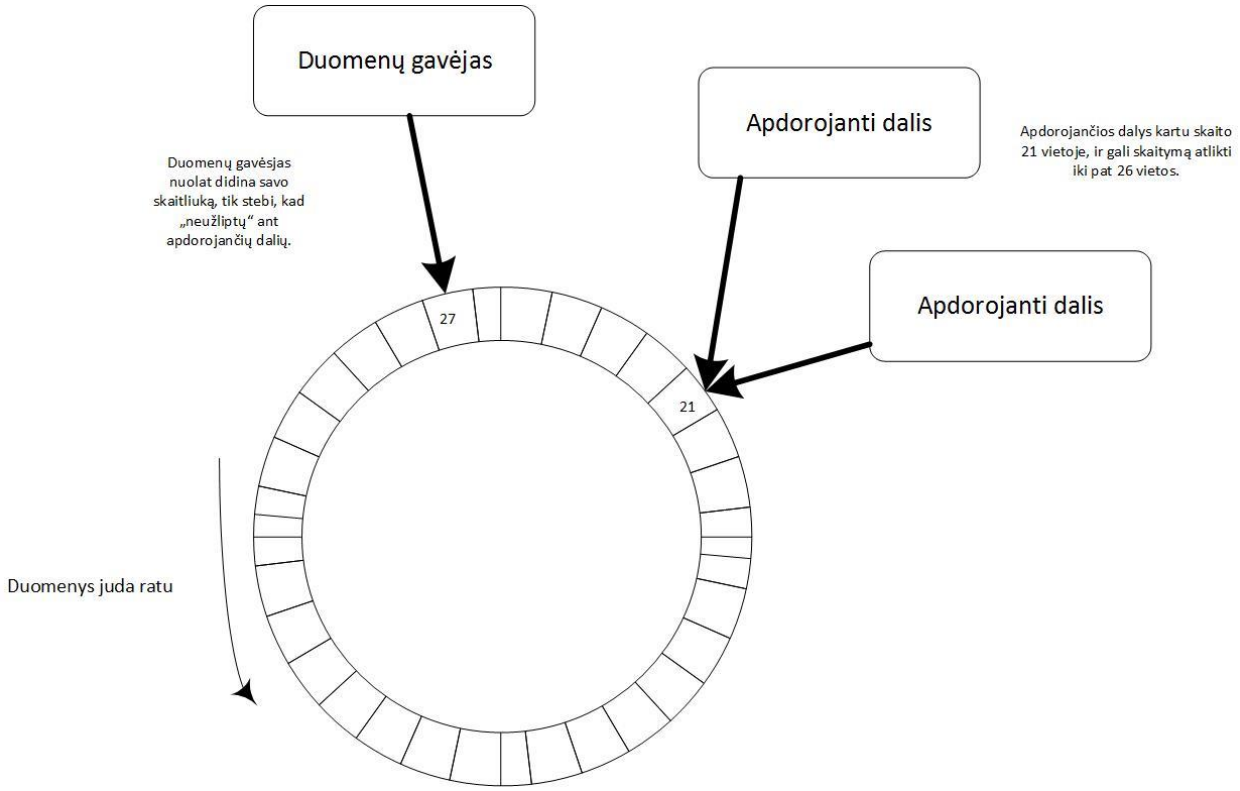
Taip pat „Disruptor“ šablono realizacijoje galima naudoti skirtingas realizacijas, kuriomis užlaikomos skirtingos gijos dalyvaujančio žiede. Galimi variantai:

„YieldingWait“ laukimo strategija – gijos užlaikomos naudojantis Thread.Sleep(0). Ši strategija yra optimali, siekiant suderinti procesoriaus resursus ir optimalų uždelsimą.

„BlockingWait“ laukimo strategija – ši strategija naudoja užraktus ir stengiasi neapkrauti stipriai procesoriaus, dėl kurio nukenčia uždelsimas ir sparta.

„BusySpinWait“ laukimo strategija – ši strategija naudoja daugiausiai sistemos resursų, bet tuo pačiu pateikia ir geriausius rezultatus. Strategija tinkama naudoti, kai apdorojančių dalių kiekis mažesnis nei procesoriaus branduolių skaičius.

„SleepingWait“ laukimo strategija – ši strategija, kaip ir „BlockingWait“ strategija, stengiasi neapkrauti procesoriaus, tačiau nukenčia uždelsimas, todėl strategija tinkama tada, kai nereikalinga sparti reakcija į įvykius.



29 pav. “Disruptor” šablonas

Į darbą šį metodą įtraukiau dėl to, kad tai anot kūrėjų sparčiausias laisvai prieinamas modelis, leidžiantis išdalinti duomenis skirtingoms gijoms, jį galima sukonfigūruoti atsižvelgiant į skirtingus procesoriaus resursų ir atminties poreikius.

7. Realizavimas

7.1 Eksperimentų modelio įgyvendinimas

Įgyvendinant įvairius duomenų dalijimo modelius su didesnėmis problemomis nesusidurta, visi modeliai gan plačiai taikomi, dažniausiai nesigilinant į jų panaudojimo paskirtį ar spartą, kodo pavyzdžių taip pat galima rasti įvairiomis kalbomis. Daugiau problemų sukėlė „Disruptor“ šablono panaudojimas, nes kūrėjai kūrė jį ant JAVA platformos, ir kitų kalbų nekūrė ir nepalaiko, yra tik trečiųjų šalių perrašyti variantai į kitas programavimo kalbas. Taip pat iškilo problemų gauti reikiamą techninę įrangą bandymams atlikti. Programinė įranga buvo sukurta ant nešiojamo kompiuterio, kurio procesorius turi 2 fizinius procesorius ir dar 2 loginius procesorius, tokie procesoriaus parametrai neleistų tinkamai patikrinti plečiamumo galimybių, darbiniame kompiuteryje procesorius turi 8 fizinius branduolius, tačiau nebuvo galimybės sukompiliuoti .NET kodą, o leisti sukompiliuotas programas iš trečiųjų šalių neleidžia vidinės taisyklės. Buvo nuspręsta išsinuomoti dedikuotą serverį su 24 fiziniais branduoliais, tačiau įmonė sutiko leisti savaitę išbandyti nemokamai. Eksperimentų programa buvo pilnai automatizuota, nuo paleidimo iki rezultatų ataskaitos sudarymo. Programos vykdymas truko apie parą laiko.

7.2 Programiniai sprendimai

Bandymų realizavimas atliktas su C# programavimo kalba ir ne žemesne nei .NET 4 platforma. C# pasirinkta todėl, kad darbo vadovas jau turi sukurtą ir tobulinimą automatizuotą prekybos agentą, tad atrastus sprendimus bus galima pritaikyti praktiškai. Ne žemesnė nei 4 versijos .NET platforma pasirinkta todėl, kad su 4-tąja versija buvo pristatyta biblioteka, palengvinanti darbą su procesų lygiagretinimu. Programinių sprendimų realizavimui C# programavimo kalba naudojausi legalia Visual Studio 2013 Professional versija, kurios licenciją suteikia universitetas. Professional versijos, skirtingai nuo nemokamų versijų, privalumas tas, kad joje yra įrankiai, leidžiantys analizuoti programinės įrangos spartą, ir taip nustatyti taškus, kurie labiausiai lėtina kuriamos sistemos darbą, mano atveju tai padėjo išaiškinti metodų spartumo ar lėtumo priežastis realizuojant skirtingus modelius.

7.2.1 .NET 4

Prieš tai buvusiose .NET karkaso versijose, programuotojams buvo sunkiau išnaudoti daugiabranduolinio procesoriaus teikiamus privalumus. Reikėdavo kurti, kontroliuoti ir

sinchronizuoti atskiras gijas naudojant sudėtingus metodus kurie ne visada yra efektyvus ir geba išnaudoti visas modernių daugiabranduolių procesorių teikiamas galimybes.

Su nauja .NET 4 versija buvo pristatyta ir nauja lygiagrečių skaičiavimų biblioteka (ang. *Task Parallel Library – TPL*), kuri buvo sukurta šiuolaikinių procesorių amžiuje. Biblioteka paruošta darbui su paprastomis užduotimis, reikalaujančiomis išskirstytų skaičiavimų.

Biblioteka palaiko duomenų lygiagretinimą ir užduočių lygiagretinimą, kas programų kūrėjams leidžia dirbti su skirtingomis išlygiagretinimo užduotimis nekuriant sudėtingai valdomų gijų.

Kūrėjai tvirtina, kad tai labai spartus metodas lygiagretinimui įgyvendinti, nes automatiškai parenkami geriausi metodai ir struktrūros, todėl šį metodą patyrinėsiu savo darbe.

7.3 Tyrimo strategija ir geriausio metodo išrinkimas

Bandymai atliekami simuliuojant naujų duomenų gavimą, ir skirtingais metodais jį perduodant kitoms agento dalims (apdorojančioms dalims kurias žymėsiu AD), atsakingoms už analizės informacijos paruošimą. Kiekvieno metodo apdorojančios dalys gavusios naujus duomenis, turės perskaičiuoti slenkančio vidurkio indikatorių 4 skirtingiems periodams ir lango dydžiams:

- Periodas 1, lango dydis 1 – siekiant imituoti tik duomenų gavimą, nereikalaujantį perskaičiavimo.
- Periodas 2, lango dydis 10 – siekiant imituoti nedidelį apkrovimą, gavus naujus duomenis
- Periodas 5, lango dydis 50 – siekiant imituoti vidutinį apkrovimą, gavus naujus duomenis
- Periodas 10, lango dydis 100 – siekiant imituoti didelį apkrovimą, gavus naujus duomenis

Siekiant detalesnių rezultatų periodo ir lango reikšmes galima keisti, tačiau mano pasirinktos 4 reikšmės apima pakankamą kiekį duomenų iš kurių galima susidaryti išvadas apie skirtingus metodus. Taip pat tokio dydžio periodai naudojami ir realioje prekyboje, tiek asmenų, atliekančių techninę analizę, tiek ir automatizuoto prekybos agento atveju.

Kiekvienam metodui eksperimento būdu bus parinktas toks kiekis naujų duomenų, kad vykdymas užtruktų bent 10 sekundžių ir neilgiau minutės, siekiant apsidrausti nuo lėtai veikiančių metodų. 10 sekundžių kiekis buvo pasirinktas dėl to, kad renkantis mažiau, tarp skirtingų matavimų, apdorotų duomenų kiekiai skirdavo pakankamai nemažai, o skaičiavimams vykstant 10 sekundžių ir ilgiau, skirtumai minimalūs. Kiekis eksperimento būdu parenkamas todėl, kad

procesorius nebūtų papildomai apkraunamas laiko skaičiavimu, todėl laiko reikšmės bus nuskaitomos tik prieš paleidžiant vykdymą ir jį pilnai baigus. Turint vykdymo laiką ir apdorotų įvykių skaičių, suskaičiuojama sparta įvykiais per skundę. Siekiant kuo mažesnės operacinės sistemos ir kitų programų įtakos rezultatams, kiekvieno metodo vykdymas pakartojamas 4 kartus, ir išvedamas spartos vidurkis. Bandymų skaičių galima didinti, tačiau eksperimentų būdu patikrinta, kad didelės įtakos vidurkiui tai nesudaro, tačiau laiko atžvilgiu užtruktų gerokai ilgiau, nei teikiama nauda. Taip pat siekiant įvertinti sistemos plečiamumą didėjant procesoriaus branduolių skaičiui, bandymai atliekami su skirtingu skaičiumi apdorojančiųjų agento. Siekiant kad apdorojančios agento dalys nedarytų įtakos rezultatams, visos apdorojančios dalys atlieka vienodas operacijas. Duomenys automatiškai sugeneruotoje ataskaitoje registruojami tokiu šablonu:

1 lentelė. Duomenų rinkimo šablonas

Periodas x Lango dydis y					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis					
Laikas (s)					
Sparta (vnt/s)					
AD					

Geresnei duomenų vizualizacijai vėliau taip pat rankiniu būdu generuojami ir grafikai.
Geriausias metodas bus tas, kuris turės didžiausią spartą.

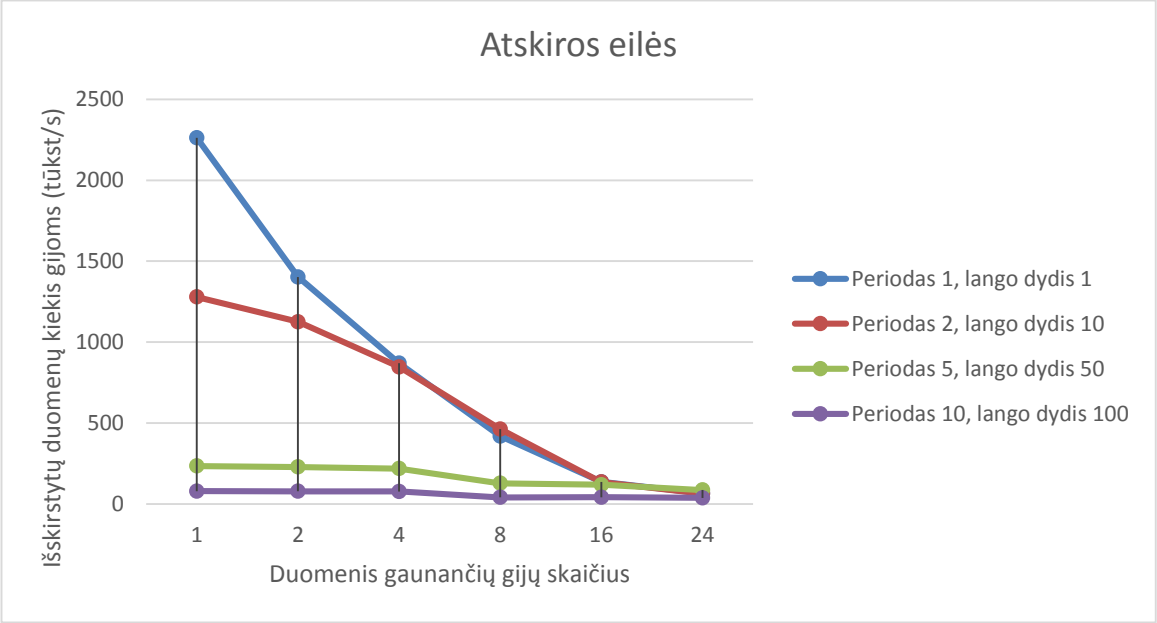
7.4 Rezultatų ataskaita

7.4.1 Atskira „concurrent“ eilė kiekvienai apdorojančiai daliai

Šis metodas antras pagal greitumą ir įvairiuose šaltiniuose nurodomas kaip greičiausias metodas išdalinti duomenis išlygiagretintiems procesams. Su duomenimis, kuriems reikalingas mažas apdorojimas, metodas rodė antrą rezultatą ir spartos pranašumas nuo prasčiausiai pasirodžiusių metodų buvo didelis, tačiau rezultatas nuo geriausiai pasirodžiusio metodo taip pat skyrėsi daugiau nei dvigubai, tačiau su labiau apkrovimo reikalaujančiu apdorojimu šis metodas rodė nežymiai geresnį rezultatą nei kiti. Labiausiai apkraunama vieta, eilės, į kurias įrašomi nauji duomenys ir vėliau išimami. Bandymams buvo panaudota .NET4 karkase realizuota eilė, kuri naudoja „SpinWait“ algoritmą, kad apsaugotų duomenų bloką, nuo skirtingų gijų įrašymo vienu metu, galimas analogiškos eilės realizavimas su užraktais.

7.4.1.1 Plečiamumas ir efektyvumas

Iš grafiko 30 pav, galime matyti, kad esant 4 ir daugiau gijų, su minimaliu ir nedideliu (periodas 1 ir lango dydis 1 bei periodas 2 ir lango dydis 10) procesoriaus apkrovimu sparta susivienodina, tokie rezultatai parodo, kad duomenų išdalijimas nėra labai efektyvus, ir galima daryti prielaidą kad procesorius arba apkraunamas nereikalingu darbu arba jį užlaiko programiniai užraktai, kurie ir suvienodina spartą.



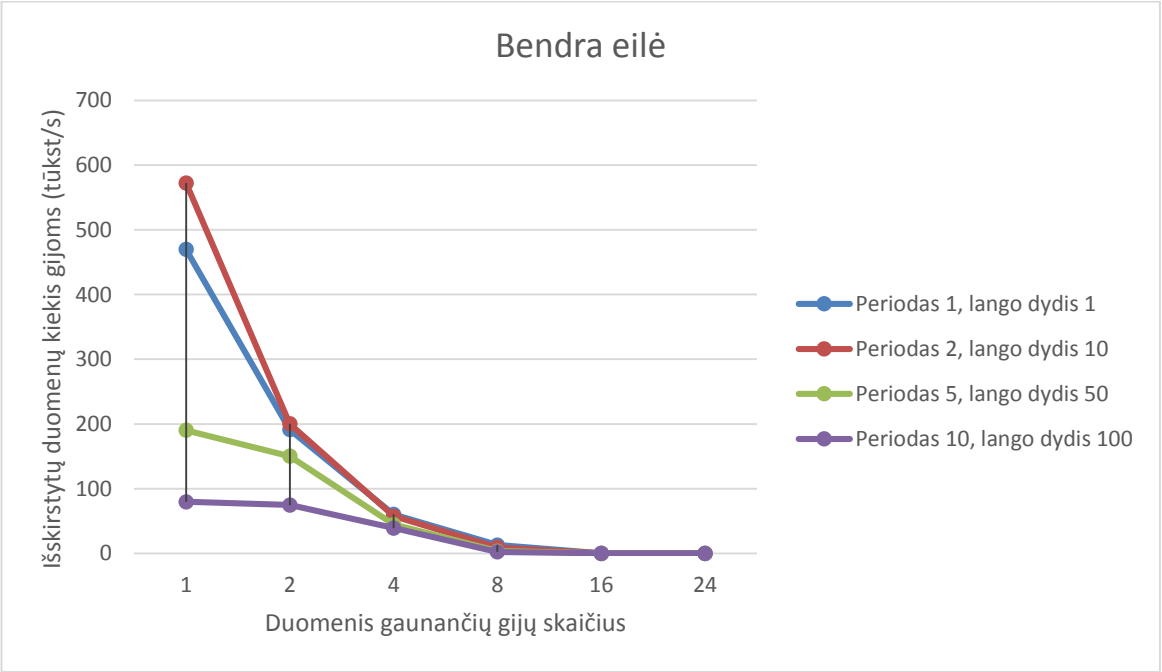
30 pav. sparta atskirų eilių metode

7.4.2 Bendra „concurrent“ eilė su užraktais

Šis metodas parodė vieną iš prasčiausių rezultatų. Taip pat metodas buvo testuotas tik su aštuoniomis gijomis, nes didinat apdorojančių dalių kiekį veikdavo labai ilgai (kelias valandas), todėl buvo sunku rasti optimalų operacijų skaičių.. Didėjant apdorojančių dalių skaičiui, sparta mažėjo, ir daugeliu atvejų buvo prasčiausia tarp visų metodų. Priežastis per didelis kreipinių skaičius į eilę, kurį naudoja visos apdorojančios dalys, duomenų gavėjas ir dar valytojas, taip pat, kiekvieną kartą pasiimant elementą iš eilės, tuo pačiu dar turi būti padidinamas skaitliukas, kuris realizuotas su užraktais.

7.4.2.1 Plečiamumas ir efektyvumas

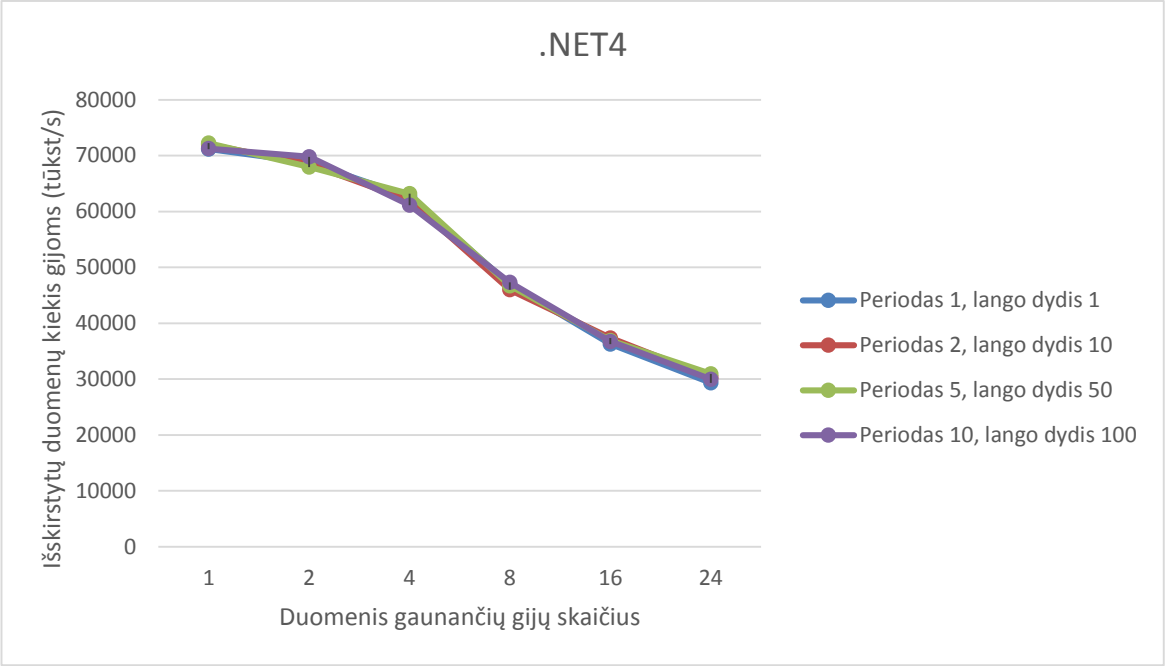
Iš grafiko 31 pav, galime matyti, kad net esant 2 gijoms su minimaliu ir nedideliu (periodas 1 ir lango dydis 1 bei periodas 2 ir lango dydis 10) procesoriaus apkrovimu sparta susivienodina, o esant 4 gijom ir daugiau, visų bandymų apkrovimų rezultatai labai panašūs, nėra didelio atotrūkio, tokie rezultatai parodo, kad duomenų išdalijimas nėra efektyvus, ir galima daryti prielaidą kad procesorius arba apkraunamas nereikalingu darbu arba jį užlaiko programiniai užraktai, kurie ir suvienodina spartą.



31 pav. sparta bendros eilės metode

7.4.3 .NET4 lygiagretinimo įrankiai

Šis metodas taip pat parodė vieną iš prasčiausių rezultatų. Ypatingai prasti rezultatai buvo, kur naujiems duomenimis nereikalingas apdorojimas. Bandymų metu stebint procesoriaus apkrovimą, buvo stebima, kad procesorius nėra apkraunamas su apdorojimo nereikalaujančiu duomenų apdorojimu, apkrovimas didėjo tik didėjant apdorojimo reiklumui (didinant slenkančio vidurkio periodą ir langą) tuo pačiu didėjo ir spartos rezultatas. Prasto rezultato priežastis, išlygiagretintų užduočių paleidimo laikas. Įsigilinus į literatūrą ir remiantis bandymų rezultatais, naudojant Parrallel.ForEach prieš pradedant vykdymą yra paskirstomi duomenys atskiroms gijoms, siekiant vykdymą atlikti kiek įmanoma sparčiai, todėl šis metodas nėra tinkamas trumpai trunkančiam apdorojimui.



32 pav. sparta .NET4 lygiagretinimo bibliotekoje

7.4.4 „Disruptor“ šablonas

Vertinant bendrai ir nesigilinant į šio metodo naudojamas strategijoas, šis metodas bandymuose parodė geriausius rezultatus. Su daug apdorojimo nereikalaujančiais įvykiais sparta buvo daugiau nei dvigubai didesnė, nei antrą geriausią rezultatą parodžiusio metodo. Didėjant apdorojimo sunkumui, spartos pranašumas mažėjo (didesnį laiko dalį užimdavo apdorojimas nei pačio įvykio perdavimas), o prie didžiausių apdorojimo sunkumo, sparta buvo nežymiai mažesnė nei antrą geriausią rezultatą rodžiusio metodo aprašyto 6.1 skyriuje. Atliekant bandymus buvo pastebėta, kad spartą stipriai įtakoja žiedinio buferio dydis ir nežymiai pasirinkta procesoriaus laukimo strategija. Geriau įsigilinus į kūrėjų dokumentaciją, atrasta, kad optimaliausia kai žiedinio buferio skaičius viršija apdorojamų duomenų skaičių, o idealiausia, jei žiedinio buferio dydis tuo pačiu telpa ir į procesoriaus trečiojo lygio spartinančiąją atmintinę (L3 cache). Mano bandymų metu, žiedinis buferis buvo sudarytas iš 134217728 elementų, ir L3 atmintinės dydį viršijo (testuojamas procesorius turėjo 25MB L3 spartinančiosios atminties), tačiau realiose sąlygose toks žiedinio buferio dydis nebūtų pasiekiamas, todėl žiedinio buferio dydis gali būti mažesnis.

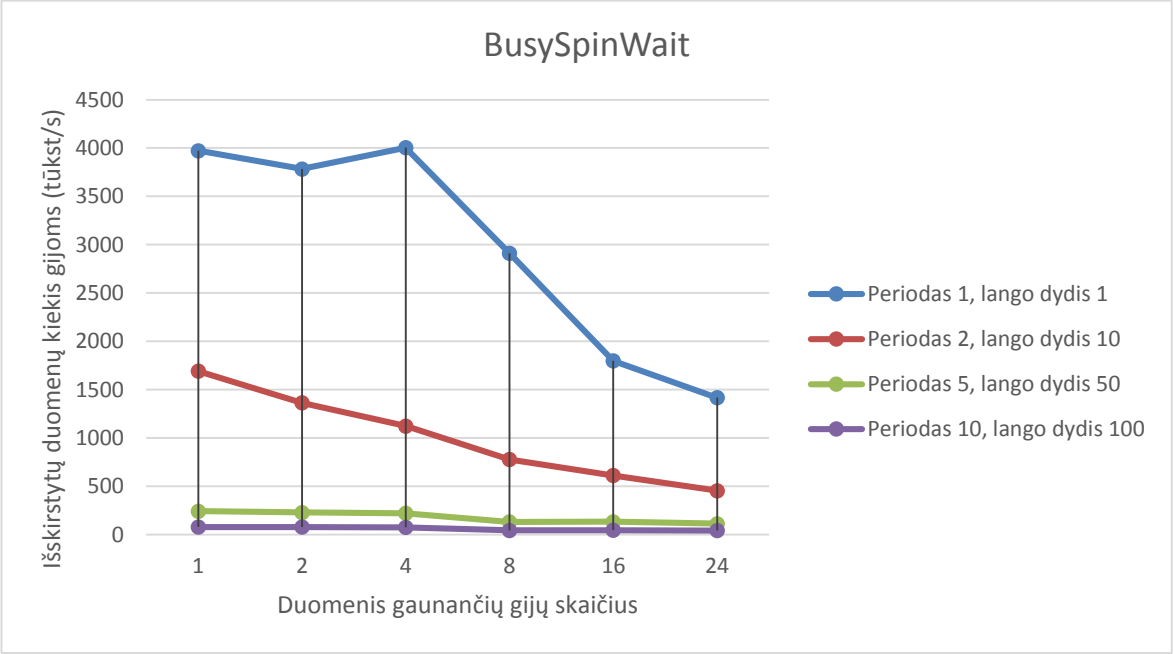
„Disruptor“ šablonas buvo išbandytas su visomis siūlomomis laukimo strategijomis.

7.4.4.1 BusySpinWait

„BusySpinWait“ laukimo strategija – ši strategija naudoja daugiausiai sistemos resursų, bet tuo pačiu pateikia ir geriausius rezultatus. Tiek bendrame rezultatų kontekste tiek ir tarp „disruptor“ šablono rezultatų, ši laukimo strategija parodė geriausią rezultatą, ir tik su 4 gijom rezultatas buvo labai panašus „sleepingWait“ laukimo strategijos rezultatą.

7.4.4.1.1 Plečiamumas ir efektyvumas

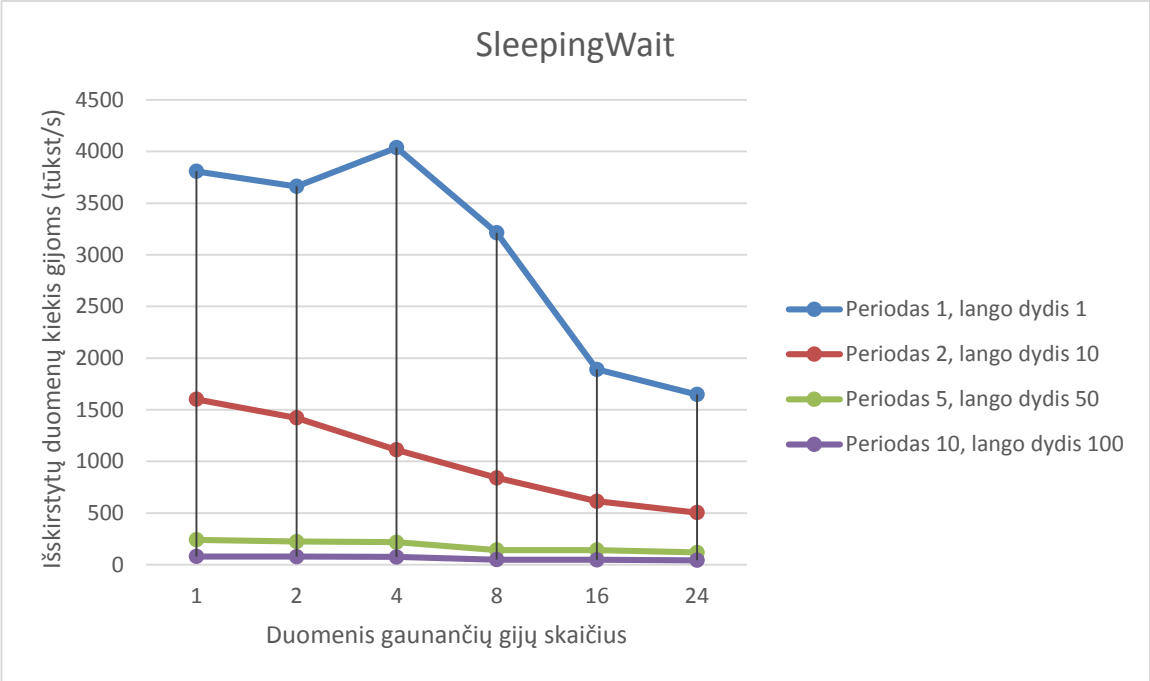
Skirtingai nei prieš tai aprašyti duomenų dalijimo modeliai, šis modelis duomenis tarp gijų išdalina efektyviai, ir iš 33 pav. puikiai matyti, kad nei su vienu bandytu apkrovimu, rezultatai netampa panašūs. Kodėl rezultatas su 4 gijom geresnis nei su 2, išsiaiškinti nepavyko, tačiau eksperimentą pakartojus net kelis kartus, rezultatas visad būdavo toks pats.



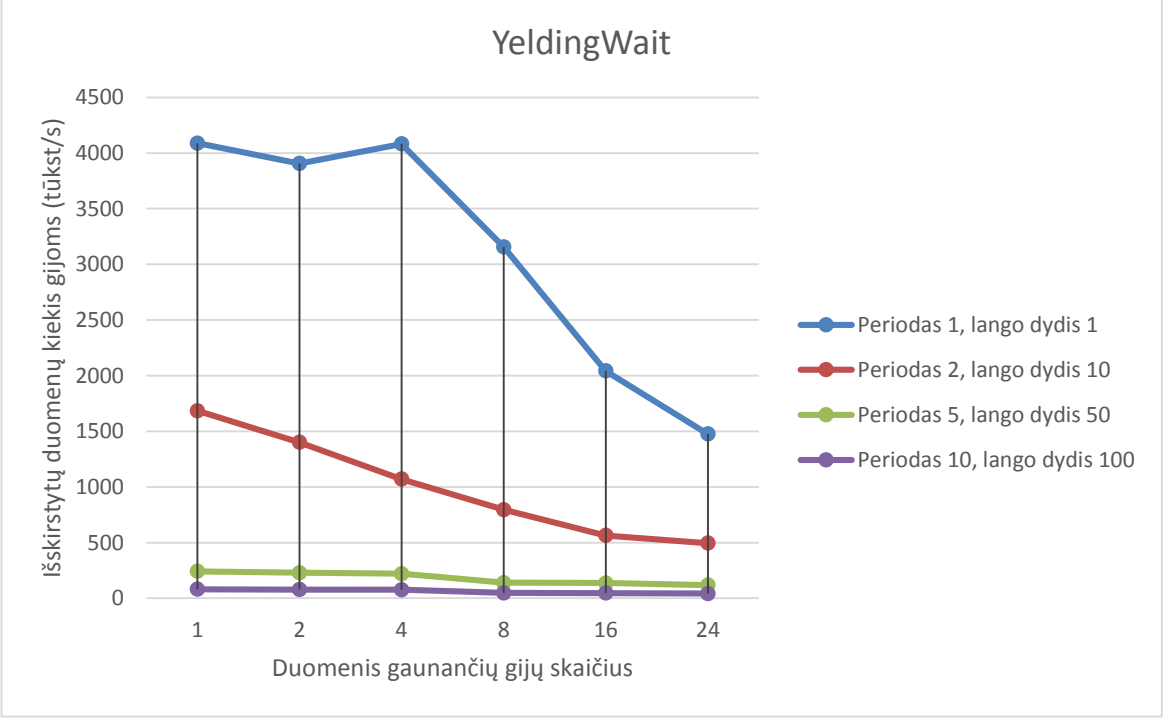
33 pav. sparta BusySpinWait laukimo strategijoje

7.4.4.2 Sleeping wait ir Yelding wait

Strategijos parodė labai panašius rezultatus, plečiamumas ir efektyvumas toks pats kaip ir BusySpinWait strategijos tik sparta mažesnė.



34 pav. sparta SleepingWait laukimo strategijoje



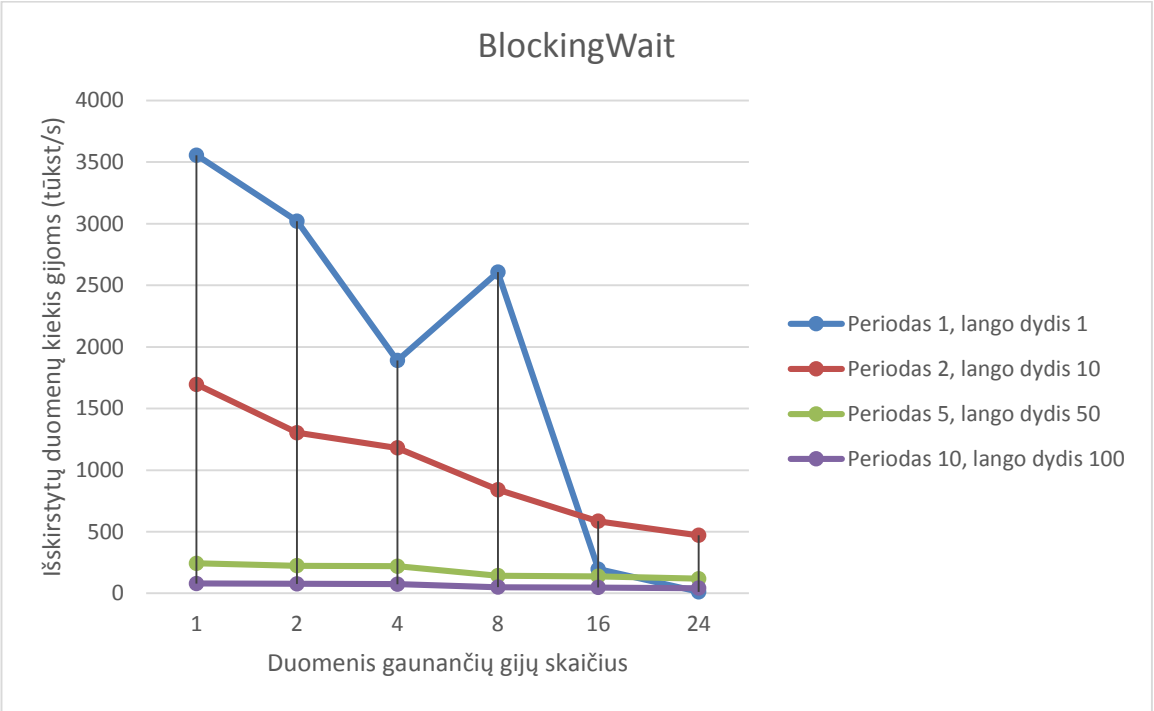
35 pav. sparta YeldingWait strategijoje

7.4.4.3 Blocking wait

Ši strategija pasirodė prasčiausiai iš visų “disruptor” šablono strategijų, taip pat su itin mažu apkrovimo, sparta buvo nepastovi.

7.4.4.3.1 Plečiamumas ir efektyvumas

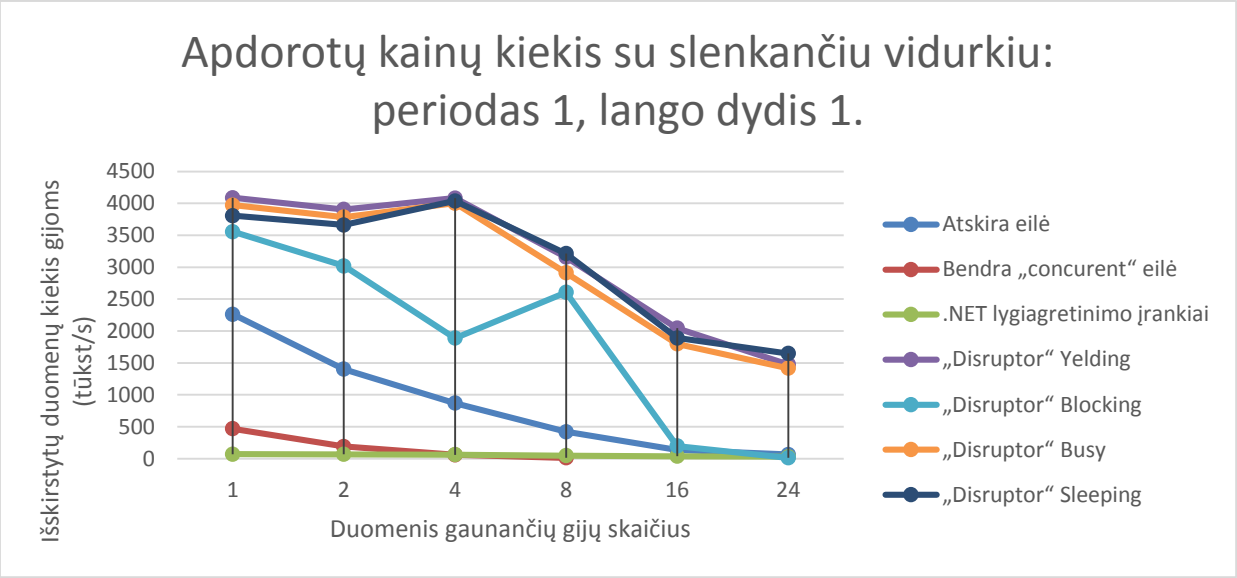
Pakartojus bandymą keletą kartų rezultatai vis tiek buvo panašūs. Galima daryti prielaidą, kad ši strategija nėra reikli procesoriaus resursams, nes naudoja operacinės sistemos užraktus, tačiau, susiklosčius blogoms sąlygoms procesoriaus instrukcijų valdyme, dėl itin mažo apkrovimo gijos užlaikomos per ilgai, todėl mažos apimties užduotims šis metodas netinkamas.



36 pav. sparta BlockingWait laukimo strategijoje

7.4.5 Periodas 1, langas 1

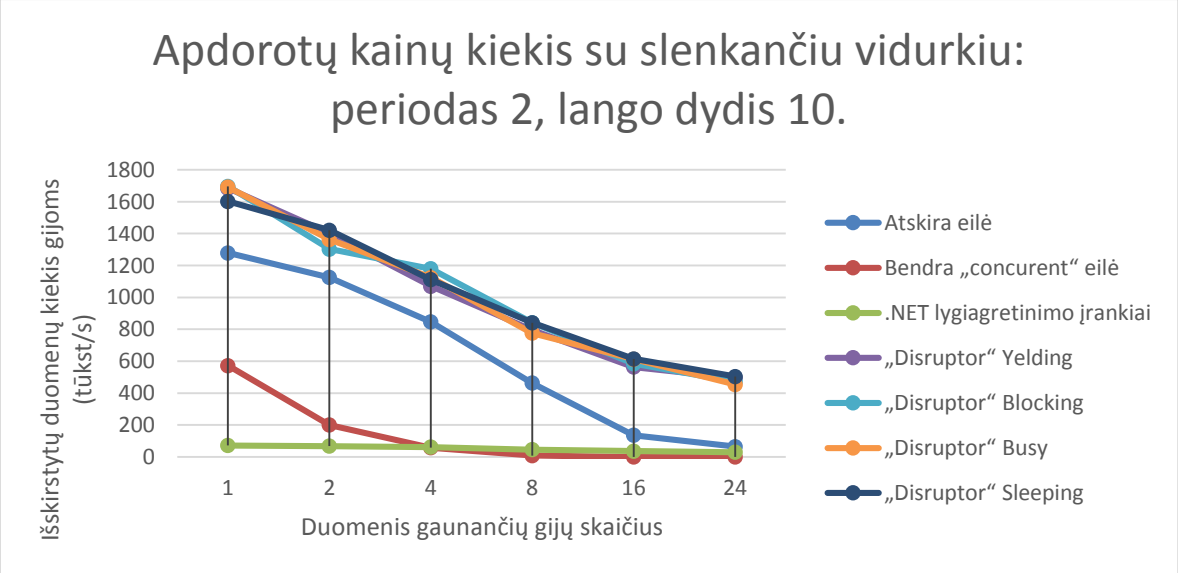
Subendrinus rezultatus slenkančiam vidurkiui su periodu 1 ir lango dydžiu 1, kuriam nereikėjo skaičiavimo resursų, sparčiausias buvo „disruptor“ šablonas naudojantis „YeldingWait strategiją“. Nors tas pats „disruptor“ šablonas su „SleepinWait“ strategija parodė geresnį rezultatą su 8 apdrojančiomis dalimis.



37 pav. sparta su periodo dydžiu 1, langa dydžiu 1

7.4.6 Periodas 2 langas 10

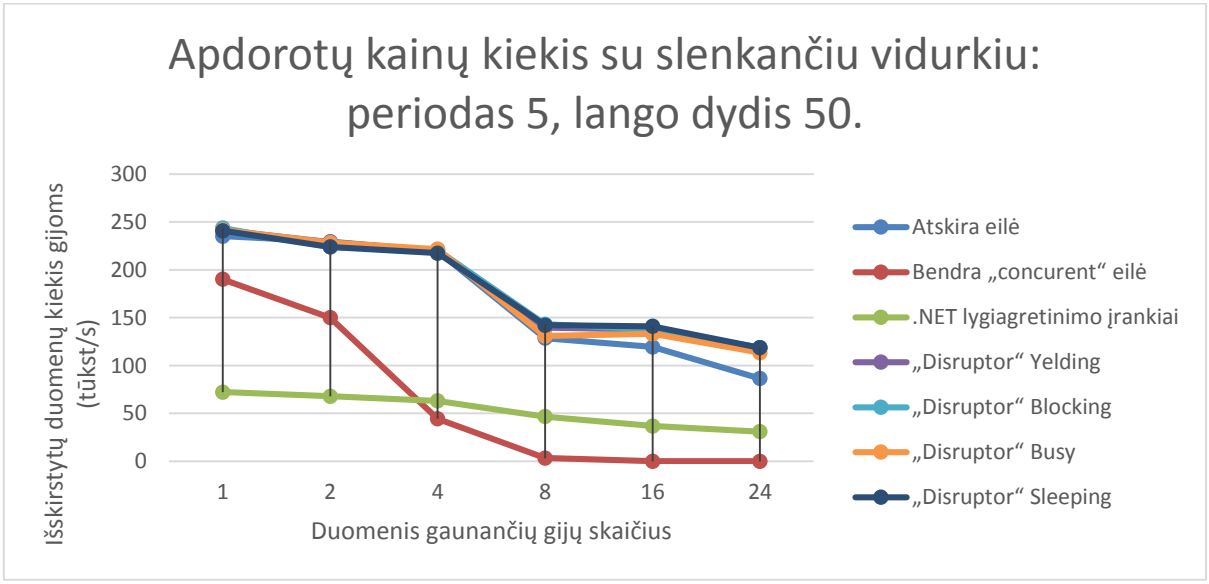
Subendrinus rezultatus slenkančiam vidurkiui su periodu 2 ir langa dydžiu 10, kuriam reikėjo minimalių skaičiavimo resursų, sparčiausi buvo „disruptor“ šablonai naudojantys „BlockingWait strategiją“, ir „SleepingWait“ strategiją. Pirmoji labiau tinkama esant apdorojančių dalių skaičiui iki 8 „SleepingWait“ geresnį rezultatą rodė esant gijų skaičiui didesniam nei 8.



38 pav. sparta su periodo dydžiu 2, langa dydžiu 10

7.4.7 Periodas 5 langas 50

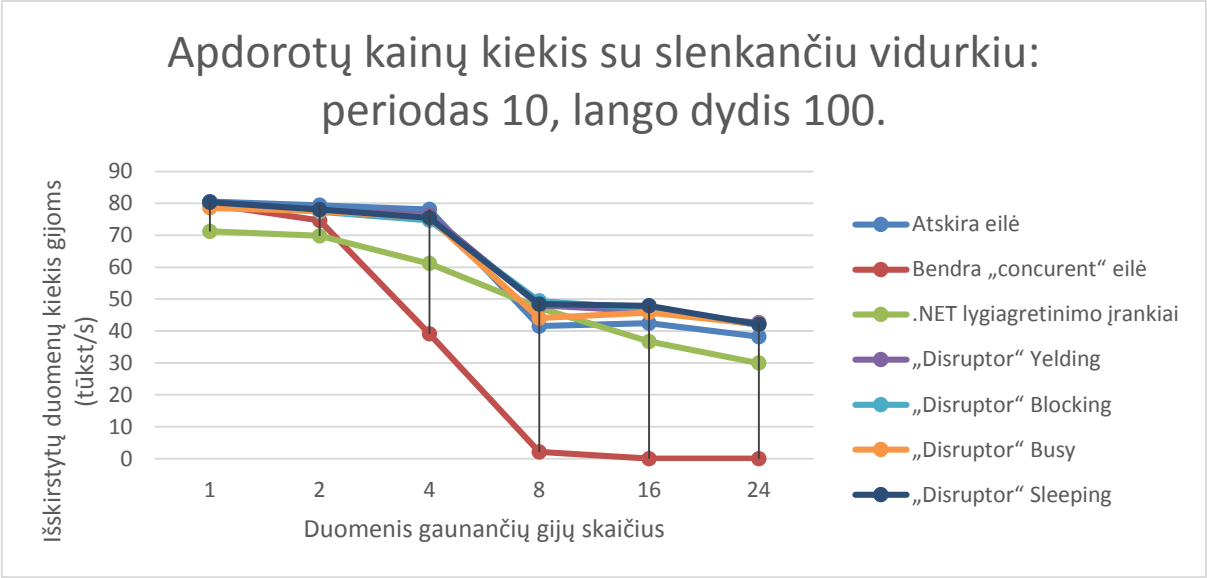
Subendrinus rezultatus slenkančiam vidurkiui su periodu 5 ir lango dydžiu 50, kuriam reikalingi vidutiniai skaičiavimo resursai, panašūs rezultatai su nedideliais skirtumais buvo net penkių metodų: atskirų eilių kiekvienai apdorjančiai daliai, „disruptor“ su visomis keturiomis strategijomis. Geriausiai pasirodė disruptor šablonas, bet didesnių skirtumų kurią srategiją naudoti. Astkirų eilių metodas rodė panašų rezultatą iki 16 apdorojančių dalių, didinat šių dalių kiekį, rezultatai buvo kiek prastesni nei disruptor.



39 pav. sparta su periodo dydžiu 5, lango dydžiu 50

7.4.8 Periodas 10 langas 100

Subendrinus rezultatus slenkančiam vidurkiui su periodu 10 ir lango dydžiu 100, kuriam reikalingi dideli skaičiavimo resursai, panašūs rezultatai su nedideliais skirtumais, kaip ir prieš tai buvusio metodo atveju, buvo net penkių metodų: atskirų eilių kiekvienai apdorojančiai daliai, „disruptor“ keturios strategijos. Nedideliu skirtumu nuo geriausiųjų atsiliko ir .NET4 lygiagretinimo įrankiai, o su 8 apdorojančiom dalimis rezultatas ketvirtas geriausias. Iki 4 apdorojančių dalių geriausiai pasirodė metodas, naudojantis atskiras eiles kiekvienai apdorojančiai daliai. Didėjant apdorojančių dalių kiekiui, geriausiai susitvarkė „disruptor“ šablonas su sleepingWait strategija.



40 pav. sparta su periodo dydžiu 10, lango dydžiu 100

7.5 Apibendrinimas

Iš viso buvo atlikta 2688 bandymai, kurių vykdamas užtruko apie 12 valandų. Visi metodai buvo palyginti apskaičiuojant slenkantį vidurkį, naudojant keturis skirtingus parametrų rinkinius: periodas 1, lango dydis 1; periodas 2, lango dydis 10; periodas 5, lango dydis 50; periodas 10, lango dydis 100.

- Su slenkančio vidurkio periodu 1 ir lango dydžiu 1 (duomenys nereikalaujantys apdorojimo) geriausią rezultatą parodė „disruptor“ šablonas su „yeldingWait“ laukimo strategija.
- Su slenkančio vidurkio periodu 2 ir lango dydžiu 10 (duomenys reikalaujantys nedidelio apdorojimo) geriausią rezultatą parodė „disruptor“ šablonai su „blockingWait“ ir „sleepingWait laukimo strategijomis. „BlockingWait“ geresnį rezultatą rodė iki 8 apdorojančių dalių, esant daugiau apdorojančių dalių, geresnis rezultatas buvo „sleepingWait“ strategijos.
- Su slenkančio vidurkio periodu 5 ir lango dydžiu 50 (duomenys reikalaujantys vidutinio apdorojimo) geriausią rezultatą rodė net penkti metodai: „disruptor“ šablonas su visomis laukimo strategijomis, ir atskirų eilių metodas, tačiau paskutiniojo metodo rezultatai suprastėjo esant apdorojančių dalių skaičiui didesniam nei 8 ir nuo „disruptor“ šablono jis atsiliko.

- Su slenkančio vidurkio periodu 10 ir lango dydžiu 100 (duomenys reikalaujantys didelio apdorojimo) iki 8 apdorojančių dalių geriausiai pasirodė metodas su atskiromis eilėmis, didėjant apdorojančių dalių skaičiui, geriausią rezultatą parodė „disruptor“ su „sleepingWait“ laukimo strategija.
- .NET4 lygiagretinimo įrankiai juntamą naudą duotų tik tada, kai reikalingas didelis apdorojimas, didesnis nei buvo bandyta.
- Sparčiausias lengviausiai įgyvendinamas sprendimas – atskiros „concurrent“ eilės.
- Universaliausias metodas, visiems atvejams, parodęs geriausius rezultatus – „disruptor“ šablonas.
- Siekiant geriausio rezultato „disruptor“ šablono laukimo strategija kiekvienam panaudojimo atvejui turėtų būti pasirinkta unikaliai.

Rezultatai ir išvados

Internete yra nemažai informacijos apie pavienius duomenų dalijimo modelius gijoms, tačiau dažniausiai palyginama tik su kitu populiariu metodu, nėra aiškios strategijos, kaip palyginimas atliktas, neatsižvelgiama kaip apkraunamas procesorius, mano atliktame darbe sukurta strategija ir programinė įranga, leidžia nesunkiai išmatuoti, kuris duomenų dalijimo modelis geriausiai tinka norimoje pritaikymo srityje. Į sukurtą programinę įrangą nesunku pridėti norimą palyginti algoritmą. Matavimus galima atlikti jau su realizuotais metodais, arba nesunkiai prisidėti savo.

Išvados:

- Tiriant greitų įvykių apdorojimo mechanizmus nustatyta, kad populiariausia duomenų struktūra realizuoti greitiems įvykių apdorojimo mechanizmas, yra eilė, skiriasi tik manipuliacijos duomenimis ir užraktai, apsaugai nuo skirtingų gijų įrašymo vienu metu.
- Kuriant automatizuotos prekybos agento prototipą nustatyta, kad architektūra nesiskiria nuo architektūrų skirtų darbų išlygiagretinimui, todėl strategija ir programinė įranga bandymams gali būti panaudota lygiagretinimo metodams lyginti.
- Lyginant skirtingų dalinimo mechanizmų rezultatus algoritminei prekybai, nustatyta, kad daugeliu atvejų geriausius rezultatus rodė „disruptor“ šablonas.
- Analizuojant rezultatus ir lyginant plečiamumo galimybės, nustatyta kad geriausiai plečiamas metodas „.NET4 lygiagretinimo biblioteka“.

8.1 Galimi patobulinimai

Mano tyrinėti duomenų dalinimo modeliai neįtraukia komercinių, mokamų produktų į palyginimus. Gavus priėjimą prie šių modelių, juos būtų galima įtraukti į darbą, ir taip papildyti bandytų modelių įvairovę.

Siekiant didesnės apimties rezultatų, strategiją būtų galima papildyti nurodymais, kaip atlikti ir vertinti rezultatus tarp skirtingų procesoriaus parametrų ir architektūrų:

- Mano darbe atlikti bandymai rėmėsi procesoriaus resursais, kurio panaudojimo paskirtis – x86_64 architektūros serveris skirtas didelės apkrovos servisams, turintis 24 fizinius branduolius ir didelės talpos L3 spartinančiąją atmintinę, dėl pastarosios dydžio priklauso ir darbe lyginto modelio „disruptor“ žiedinio buferio dydis, kuris įtakoja šio modelio

spartos galimybės, todėl yra galimybė, kad procesorius su maža L3 spartinančiąja atmintine ar be jos, nebūtų sparčiausias.

- Rinkoje labai sparčiai išpopuliarėją išmanieji įrenginiai taip pat atveria įvairias pritaikymo sritis kuriose greitas ir efektyvus įvykių apdorojimas svarbus ne tik dėl spartos, bet ir procesoriaus efektyvaus išnaudojimo siekiant taupyti akumuliatoriaus energiją. Beveik visi tokio tipo įrenginiai naudoja ARM architektūros procesorius, skirtingose architektūrose instrukcijų, spartinančiosios atminties organizavimas ar pertraukimai gali veikti skirtingai, todėl tokių pačių metodų rezultatai gali būti nebūtinai vienodi.
- Užduotims, reikalaujančiomis didelių skaičiavimo resursų taip pat vis dažiau panaudojami vaizdo plokščių procesoriais turintys šimtus branduolių, todėl būtų galima patyrinėtų tokių metodų panaudojimą, galimybes ir realizavimą su vaizdo plokščių procesoriais.

Literatūra

[JSS] Jakki Mohr, Sanjit Sengupta, and Stanley Slater, “Marketing of High-Technology Products and Innovations – second edition”, “Technology Map”, pg 204-211.

[Luc02] David Luckham, “The Power of Events – An Introduction to Complex Event Processing in Distributed Enterprise Systems”, Addison-Wesley 2002

[LFr98] David C. Luckham and Brian Frasca, Complex Event Processing in Distributed Systems, Program Analysis and Verification Group, Computer Systems Lab Stanford University, August 18, 1998

[GWD] Daniel Gyllstrom, Eugene Wu, Hee-Jin Chae Yanlei Diao, Patrick Stahlberg, Gordon Anderson, SASE: Complex Event Processing over Streams, Department of Computer Science University of Massachusetts, Amherst

[Ora] Jyrki Oraskari, Complex Event Processing, Department of Computer Science and Engineering, Aalto University, Espoo, Finland

[HWo] Gregor Hohpe, Bobby Wool, Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions, Addison-Wesley Professional

[Fow14] Martin Fowler, The LMAX Architecture,
[žiūrėta: 2014-06-05]. Prieiga per internetą:
<http://martinfowler.com/articles/lmax.html >

[KGR12] Ramkumar (Ram) Krishnan, Jonathan Goldstein, Alex Raizman, A Hitchhiker’s Guide to Microsoft StreamInsight Queries - (Version: Jun 12 2012)

[Syb] Sybase, Complex Event Processing: Ten Design Patterns

[WebM] Business Activity Monitoring (BAM), The New Face of BPM. WebMethods, [žiūrėta: 2014-06-05]. Prieiga per internetą: <http://www.business.unr.edu/faculty/kuechler/788/bam-the_new_face_of_bpm.pdf>

[Mic] Brenda M. Michelson, Event-Driven Architecture Overview, www.psgroup.com

[Etz] Opher Etzion, Towards an Event-Driven Architecture: An Infrastructure for Event Processing Position Paper, IBM EDA Initiative Leadership Team.

[ADG06] Adi, A. ; Botzer, D. ; Nechushtai, G. ; Sharon, G. Complex Event Processing for Financial Services, IEEE, 2006

[RGY09] Fethi A. Rabhi, Adnene Guabtni, Lawrence Yao, A data model for processing financial market and new data, Inderscience Publishers, 2009

[DRe13] Mohd. Saboor, Rajesh Rengasamy, Designing and developing complex event processing applications, 2013

[SLS] Mohammad Sadoghi, Martin Labrecque, Harsh Singh, Warren Shum, HansArno Jacobsen. Efficient Event Processing through Reconfigurable hardware for Algorithmic Trading

[Rei] James Reinders. Intel Threading Building blocks

[BGA] Roger S. Barga, Jonathan Goldstein, Mohamed Ali, Mingsheng Hong: Consistent Streaming Through Time: A Vision for Event Stream Processing

[Bas] Tim Bass. Mythbusters: Event Stream Processing v. Complex Event Processing. [žiūrėta: 2014-01-05]. Prieiga per internetą: <http://www.debs.msrg.utoronto.ca/bass.pdf>

[Obe] Supreet Oberoi. Introduction to complex event processing & data streams. [žiūrėta: 2014-01-15]. Prieiga per internetą: <http://www.rti.com/docs/SOAWM7_8_oberoi.pdf>

[Ber] Thomas Bernhardt, Esper, [žiūrėta: 2014-06-05]. Prieiga per internetą: <http://esper.codehaus.org>

[Seaa]SearchCIO [žiūrėta: 2014-06-05]. Prieiga per internetą: <http://searchcio.techtarget.com/definition/business-activity-monitoring-BAM>

[Seab]SearchSOA
[žiūrėta: 2014-06-05]. Prieiga per internetą:
<http://searchsoa.techtarget.com/definition/EAI>

Priedai

1. Priedas. Techninė įranga

Programinė įranga buvo kurta nešiojamuoju kompiuteriu kurio parametrai:

Operacinė sistema: Windows 8.1 64bit

Procesorius: Intel Core i5-2430M CPU @ 2.4GHz

Aprašymas: Description: Intel64 Family 6 Model 42 Stepping 7

Taktinis dažnis: 2395MHz

Branduolių skaičius: 2

Loginių branduolių skaičius: 4

HyperThreading: įjungtas

Operatyvioji atmintis: 8099MB

L1 spartinančioji atmintinė: 64KB

L2 spartinančioji atmintinė: 512KB

L3 spartinančioji atmintinė: 3072KB

Bandymams taip pat naudotas serverinis kompiuteris:

Operacinė sistema: Windows Server 2003

Procesorius: 2x - Intel® Xeon® Processor 5639 @ 2.13GHz

Aprašymas:

Taktinis dažnis: 2130

Branduolių skaičius: 12

Loginių branduolių skaičius: 24

HyperThreading: įjungtas

Operatyvioji atmintis: 8GB

L1 spartinančioji atmintinė: 768KB

L2 spartinančioji atmintinė: 3MB

L3 spartinančioji atmintinė: 24MB

2. Priedas. HyperThreading technologija

Hyper-Threading technologija išplečia lygiagretiškumą instrukcijų lygyje, padvigubindama architektūrines būsenas, tam kad būtų galima išnaudoti lygiagretaus kodo trūkumus, paleidžiant antrą giją, kai pirmoji veikia bet laukia kažkokio resurso. Tokiu atveju atrodo, kad procesorius turi dvigubai daugiau branduolių nei iš tikro turi fizinių branduolių. Todėl labai svarbu suprasti, kad loginiai branduoliai nėra tas pats kas realūs, fiziniai, branduoliai. Kartais ši technologija pagerina spartą, nes instrukcijų lygyje atsiranda dvi gijos su nepriklausomais instrukcijų šaltiniais. Tačiau jei programoje paleistos gijos neturi daug priklausomybių nuo duomenų, paspartėjimas gali būti ne toks, kokio tikėtasi. Apibendrinant tai labai priklauso nuo programos tipo.

3. Priedas .NET4 lygiagretinimo karkasas

3.1 System.Threading.Tasks.Parallel Class

TPL bibliotekai buvo išskirta ir nauja vardų sritis, tai „*System.Threading.Tasks*“. Tai suteikia priėjimą prie klasių, struktūrų ir „enum‘ų“ pristatytų su nauja „.NET Framework 4“ versija, kai norima dirbti su TPL biblioteka galima naudoti šią vardų sritį:

using System.Threading.Tasks;

Taip bus išvengta bereikalingo programinio kodo. Pavyzdžiui vietoj *System.Threading.Tasks.Parallel.Invoke* galima rašyti *Parallel.Invoke*. Pagrindinė klasė yra *Task,* kuri skirta asinchroninėm ir lygiagrečioms operacijoms, tačiau nebūtina dirbti tiesiogiai su klasės *Task* implementacijomis, kad sukurti lygiagrečiai veikiantį kodą. Kartais lengviau susikurti lygiagrečiai veikiančius kodus, ar kodo sritis, todėl vietoj to, kad naudotis žemesniojo lygio *Task* implementacijomis, galima naudoti metodus iš statinės klasės *Parallel* (*System.Threading.Tasks.Parallel*):

- *Parallel.For* – lygiagrečiai paleidžia nurodytą skaičių ciklo operacijų. Operacijos veikia su krūvio paskirstymu (ang. *Load balancing*), kas leidžia apkrauti procesoriaus branduolius kiek įmanoma labiau ir su kuo mažesniu prastovos laiku (ang. *Idle time*).
- *Parallel.ForEach* – veikia panašiu principu kaip ir *Parallel.For* tačiau taip pat leidžia programuotojui kontroliuoti duomenų paskirstymą išlygiagretinant.
- *Parallel.Invoke* – nurodytų užduočių (metodų) lygiagretus vykdymas.

Šie metodai labai naudingi pertvarkant jau egzistuojantį kodą, norint išnaudoti išlygiagretinimą kur jis įmanomas. Tačiau taip pat svarbu suprasti, kad tai nėra viskas taip paprasta kaip sakinį *for* pakeisti į *Parallel.For*.

3.1.1 Parallel.Invoke

Lengviausias būdas paleisti keletą metodų vienu metu lygiagrečiai, yra naudoti naują *Invoke* metodą kurį turi klasė *Parallel*. Tarkime turime keturis skirtingus metodus, ir norime juos paleisti lygiagrečiai. Tą padaryti galima naudojantis tokia eilute:

Parallel.Invoke(Metodas1, Metodas2, Metodas3, Metodas4);

Tokį pat rezultatą galima pasiekti naudojant lambda išraiškas:

```
Parallel.Invoke(() => ConvertEllipses(),
   () => ConvertRectangles(),
   () => ConvertLines(),
   () => ConvertText());
```

Galima naudoti lambda išraiškas ir anoniminius objektus, kad paleisti norimus metodus:

```
Parallel.Invoke(
   () =>
   {
      Metodas1();
      // čia gali būti papildomas kodas
   },
   () =>
   {
      Metodas2();
      // čia gali būti papildomas kodas    },
   delegate()
   {
      Metodas3();
      // čia gali būti papildomas kodas    },
   delegate()
   {
```

```
    Metodas4();
});
```

Duoti kodo pavyzdžiai iškvietus metodą *Parallel.Invoke* netęs kodo vykdymo tol, kol nebus baigti visi (duotuoju atveju keturi) metodai, kurie buvo paleisti veikti lygiagrečiai. Įvykus klaidai (ang. *Exception*) taip pat laikoma, kad metodas darbą baigė. Metodas *Parallel.Invoke* bandys paleisti visus keturis metodus taip, kad galėtų išnaudoti visus teikiamus daugiabranduolinio procesoriaus privalumus, tačiau realus lygiagretus veikimas priklausys nuo daugelio faktorių. Šiuo atveju turime keturis metodus, kas reiškia, kad reikalingi 4 procesoriaus branduoliai, kad būtų pasiektas geriausias efektas, todėl svarbu žinoti, ant kokios įrangos veikia mūsų programinis kodas.

Tačiau keturių procesoriaus branduolių turėjimas taip pat neužtikrina, kad visi keturi metodai pradės savo darbą vienu metu. Jei vienas ar keli procesoriaus branduoliai jau užimti, tai gali sukelti vykdymo pradžios uždelsimą. Taip pat labai sunku nuspėti, kokia tvarka metodai pradės darbą.

Parallel.Invoke leidžia nesunkiai paleisti keletą metodų vienu metu, nesirūpinant apie gijų kūrimą, tačiau kartais tai nėra pats tinkamiausias sprendimas ir turi savų trūkumų:

- Jei skirtingi metodai turi skirtingą veikimo laiką, tai metodas *Parallel.Invoke* baigs darbą tik tada, kai darbą pabaigs ilgiausiai veikiantis metodas, todėl prieš naudojantis svarbu įvertinti, kiek laiko užtruks metodų vykdymas, nes tinkamai to nepadarius vienas ar daugiau procesoriaus branduolių veiks be apkrovos.
- Turint 16 branduolių procesorių ir leidžiant 4 metodus, tik 4 branduoliai bus užimti, kiti 12 apkrauti nebus.
- Kaip ir kiekviename lygiagrečiame kode, gali atsirasti bendrai vartojamų resursų, dėl kurių gali kilti sunkiai surandamų programinių klaidų, tačiau tai bendras trūkumas bet kokiam, lygiagrečiai veikiančiui kodui.
- Nėra jokių garantijų, kad nurodyti metodai pasileis tokia pat tvarka, kaip buvo nurodyta iškviečiant *Parallel.Invoke* metodą, todėl tai nėra tinkama sudėtingiems skaičiavimams, kuriuose paleidimo tvarka svarbi.
- Lygiagrečiai vykdomame kode gali kilti klaidų (ang. *Exceptions)*, o jų pagavimas ir apdorojimas tokiu atveju yra kiek sudėtingesnis nei lygiagrečiai veikiančio kodo.

3.1.2 Parallel.For

Esamus ciklus naudojančius sakinį *for* galima išlygiagretinti pakeičiant jį į *Parallel.For* ir pritaikant jam reikalingus parametrus. Pavyzdys:

```
Parallel.For(1, NUM_MD5_HASHES + 1, (int i) =>
  {
    var md5M = MD5.Create();
    byte[] data =
      Encoding.Unicode.GetBytes(
      Environment.UserName + i.ToString());
    byte[] result = md5M.ComputeHash(data);
    string hexString = ConvertToHexString(result);
    // Console.WriteLine("MD5 HASH: {0}", hexString);
  });
```

Pagrindiniai metodo *Parallel.For* parametrai:

- fromInclusive – skaičius imtinai, nuo kurio prasideda ciklas. Gali būt tiek Int32 tiek ir long(Int64) tipo skaičius.
- toExclusive – tai viršutinis rėžis, iki kurio bus vykdomas ciklas. Gali būti tiek Int32 tiek long (Int64). Ciklas vykdomas iki viršutinio rėžio reikšmės, tačiau paskutinė reikšmė nėra įtraukiama, t.y. atitikmuo būtų *for(int i=fromInclusive; i<toExlusive; i++);* Labai svarbu atkreipti į tai dėmesį, nes neretai cikluose naudojama sąlyga mažiau arba lygu (<=), todėl lygiagretinant programinį kodą gali būti reikalingi nedideli pataisymai.
- Body – ciklo skaitliukas. Gali būti Int32 arba Int64.

Parallel.For gali gražinti *ParallelLoopResult* reikšmę, nes tai lygiagrečiai veikiantis kodas, o lygiagrečiai veikiantys ciklai yra daug sudėtingesnis nei nuosekliai veikiantys. Kadangi tai ne nuosekliai veikiantis kodas, negalima nustatyti ties kuria reikšme ciklas baigė savo darbą.

3.1.3 Parallel.ForEach

Kartais egzistuojančio *for* ciklo išlygiagretinimas yra sudėtinga užduotis, nes pakeitimai kode gali apkrauti kokį nors bendrai naudojamą resursą, ir taip bus prarandama išlygiagretinimo teikiama sparta. Viena iš paprastų alternatyvų, tai padalinti duomenis, kurie reikalingi apdoroti, į dalis, kurios taip galėtų veikti nepriklausomuose cikluose lygiagrečiai. Tai galima padaryti

naudojantis metodu *Parallel.ForEach*. Šis metodas turi bendrą mechanizmą, kuris leidžia apdoroti duomenis, kurie buvo apdoroti išlygiagretintu ciklu. Pavyzdžiui, galima turėti kažkokį sveikų skaičių rinkinį, kuris dėka suskirstytojo (ang. *Custom partitioner*) padalina duomenis į atskiras dalis. Vėliau šios dalys apdorojamos lygiagrečiai leidžiant ciklus per jas. Duomenis išdalinti galima naudojantis *System.Collections.Concurent.Partitioner*. Duomenys bus išdalinti į mažesnes dalis taip, kad būtų galima paleisti tiek nepriklausomai veikiančių ciklų, kiek procesorius turi branduolių. Kodas taip pat optimizuoja savo veikimą veikimo metu priklausomai nuo tuo metu galimos techninės įrangos.

System.Collections.Concurent - taip pat nauja vardų sritis .NET 4 karkase, kuri leidžia pasiekti naudingas duomenų struktūras, kurios paruoštos darbui su lygiagretiniu.

Metodas *Parallel.ForEach* turi net 20 skirtingų implementacijų (overrides). Paprasčiausios parametrai tokie:

- *Source* – duomenys, kuriuos reikia sudalinti į atskiras dalis.
- *Body* – duomenys, su kuriais atliekami veiksmai.

4. Priedas. Atskira „concurrent“ eilė kiekvienai apdorojančiai agento daliai

4.1 Viena apdorojanti dalis

2 lentelė. Sparta su viena apdorojančia dalimi

Periodas 1, langa dydis 1					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	27,118644 milijono				
Laikas (s)	11,738	12,022	12,048	12,112	
Sparta (vnt/s)	2310329	2255751	2250883	2238989	2261960
AD	1				

Periodas 2, langa dydis 10					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	20 milijonų				
Laikas (s)	15,673	14,827	15,379	16,605	
Sparta (vnt/s)	1276079	1348890	1300474	1204456	1279426
AD	1				

Periodas 5, langa dydis 50

	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	3 milijonai				
Laikas (s)	12,937	12,461	12,955	12,756	
Sparta (vnt/s)	231893	240751	231579	237435	235183
AD	1				

Periodas 10, langa dydis 100					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	1 milijonas				
Laikas (s)	12,443	12,335	12,533	12,482	
Sparta (vnt/s)	80366	81070	79789	81070	80573
AD	1				

4.2 Dvi apdorojančios dalys

3 lentelė. Sparta su dviem apdorojančiomis dalimis

Periodas 1, langa dydis 1					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	16,824395 milijono				
Laikas (s)	11,837	11,704	12,192	12,211	
Sparta (vnt/s)	1421339	1437491	1379953	1377806	1402441
AD	2				

Periodas 2, langa dydis 10					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	12 milijonų				
Laikas (s)	10,628	10,156	10,585	11,231	
Sparta (vnt/s)	1129092	1181567	1133679	1068471	1125967
AD	2				

Periodas 5, langa dydis 50					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	3 milijonai				
Laikas (s)	13,389	12,677	13,06	13,115	
Sparta (vnt/s)	224064	236649	229709	228745	229537
AD	2				

Periodas 10, langa dydis 100					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	1 milijonas				
Laikas (s)	12,498	12,477	12,464	12,906	
Sparta (vnt/s)	80012	80147	80231	77483	79395
AD	2				

4.3 Keturios apdorojančios dalys

4 lentelė. Sparta su keturiomis apdorojančiomis dalimis

Periodas 1, langa dydis 1					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis

Duomenų kiekis	12 milijonų				
Laikas (s)	13,913	13,789	13,643	13,816	
Sparta (vnt/s)	862502	870258	879571	868558	869565
AD	4				

Periodas 2, lango dydis 10					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	12 milijonų				
Laikas (s)	14,219	13,355	13,977	15,09	
Sparta (vnt/s)	843941	898553	858553	795228	846844
AD	4				

Periodas 5, lango dydis 50					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	3 milijonai				
Laikas (s)	13,907	14,316	13,086	13,345	
Sparta (vnt/s)	215718	209555	229252	224803	219402
AD	4				

Periodas 10, lango dydis 100					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,923077 milijono				
Laikas (s)	11,727	11,63	11,666	12,217	
Sparta (vnt/s)	78713	79370	79125	75556	78106
AD	4				

4.4 Aštuonios apdorojančios dalys

5 lentelė. Sparta su aštuoniomis apdorojančiomis dalimis

Periodas 1, lango dydis 1					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	5,694626 milijono				
Laikas (s)	14,42	15,21	11,456	10,083	-
Sparta (vnt/s)	394911	374400	497086	435269	420135
AD	8				

Periodas 2, lango dydis 10					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	6 milijonai				
Laikas (s)	14,018	12,119	11,774	13,873	-
Sparta (vnt/s)	428021	495090	509597	432494	463132
AD	8				

Periodas 5, langas 50					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	1,714286 milijonai				
Laikas (s)	14,101	13,192	12,311	13,641	
Sparta (vnt/s)	121571	129948	139248	125671	128690
AD	8				

Periodas 10, langas 100

	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,545455 milijonai				
Laikas (s)	13,43	12,59	13,32	13,178	
Sparta (vnt/s)	40614	43324	40950	41391	41514
AD	8				

4.5 Šešiolika apdorojančių dalių

6 lentelė. Sparta su šešiolika apdorojančių dalių

Periodas 1, langa dydis 1					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	2 milijonai				
Laikas (s)	14,499	14,727	14,33	14,684	
Sparta (vnt/s)	137940	135804	139567	136202	137263
AD	16				

Periodas 2, langa dydis 10					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	2 milijonai				
Laikas (s)	14,997	14,613	14,575	14,515	
Sparta (vnt/s)	133360	136864	137221	137788	136188
AD	16				

Periodas 5, langa dydis 50					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	6 milijonai				
Laikas (s)	12,68	12,585	12,44	12,467	
Sparta (vnt/s)	118296	119189	120578	120317	119502
AD	16				

Periodas 10, langa dydis 100					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,48 milijono				
Laikas (s)	11,112	12,784	11,058	11,141	
Sparta (vnt/s)	43196	40424	43407	43084	42460
AD	16				

4.6 Dvidešimt keturios apdorojančios dalys

7 lentelė. Sparta su dvidešimt keturiomis apdorojančiomis dalimis

Periodas 1, langa dydis 1					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	1,333333 milijono				
Laikas (s)	18,197	19,345	21,783	21,324	
Sparta (vnt/s)	73272	68923	61209	62527	66082
AD	24				

Periodas 2, langa dydis 10					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	1,090909 milijono				

Laikas (s)	15,857	17,486	16,344	17,619	
Sparta (vnt/s)	68796	62387	66746	61916	64786
AD	24				

Periodas 5, lango dydis 50					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	1,127817 milijonai				
Laikas (s)	13,996	13,02	12,709	12,33	
Sparta (vnt/s)	80581	86621	88741	91469	86585
AD	24				

Periodas 10, lango dydis 100					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,421050 milijono				
Laikas (s)	11,077	11,43	10,79	10,751	
Sparta (vnt/s)	38011	36837	39022	39163	38206
AD	24				

5. Priedas. Bendra „Conccurent“ eilės

5.1 Viena apdorojanti dalis

8 lentelė. Sparta su viena apdorojančia dalimi

Periodas 1, lango dydis 1					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	6 milijonai				
Laikas (s)	14,033	11,749	13,749	12,286	
Sparta (vnt/s)	427563	527099	436395	488360	469831
AD	1				

Periodas 2, lango dydis 10					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	7 milijonai				
Laikas (s)	12,006	11,354	12,857	12,824	
Sparta (vnt/s)	583041	616522	544450	545851	572466
AD	1				

Periodas 5, lango dydis 50					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	4 milijonai				
Laikas (s)	12,334	11,634	11,871	13,294	
Sparta (vnt/s)	324306	343819	333695	300887	325676
AD	1				

Periodas 10, lango dydis 100					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,421050 milijono				
Laikas (s)	12,443	11,887	11,603	11,818	
Sparta (vnt/s)	160732	168251	172369	169233	167646

AD	1

5.2 Dvi apdorojančios dalys

9 lentelė. Sparta su dviem apdorojančiomis dalimis

Periodas 1, langho dydis 1					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	2 milijonai				
Laikas (s)	10,724	10,214	10,326	10,55	
Sparta (vnt/s)	186497	195809	193685	189483	191368
AD	2				

Periodas 2, langho dydis 10					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	2,4 milijono				
Laikas (s)	12,607	11,682	11,763	11,944	
Sparta (vnt/s)	190370	205444	204029	200937	200195
AD	2				

Periodas 5, langho dydis 50					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	2,4 milijono				
Laikas (s)	14,152	16,328	16,953	16,846	
Sparta (vnt/s)	169587	146986	141567	142467	150151
AD	2				

Periodas 10, langho dydis 100					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	1,4 milijono				
Laikas (s)	14,258	14,468	13,609	13,751	
Sparta (vnt/s)	101810	102873	96765	98190	99909
AD	2				

5.3 Trys apdorojančios dalys

10 lentelė. Sparta su trimis apdorojančiomis dalimis

Periodas 1, langho dydis 1					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	1,2 milijono				
Laikas (s)	11,129	10,653	10,344	10,799	
Sparta (vnt/s)	107826	112834	116009	111121	111947
AD	1				

Periodas 2, langho dydis 10					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	1,3 milijono				
Laikas (s)	12,431	11,943	11,219	10,79	
Sparta (vnt/s)	104577	108850	115874	120481	112445
AD	1				

Periodas 5, langho dydis 50

	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	4 milijonai				
Laikas (s)	12,578	10,405	11,48	10,899	
Sparta (vnt/s)	79503	96107	87108	91751	88617
AD	1				

Periodas 10, langa dydis 100					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,421050 milijono				
Laikas (s)	11,89	12,557	10,888	10,498	
Sparta (vnt/s)	67283	63709	73475	76204	70167
AD	1				

6. Priedas. .NET4 lygiagretinimo įrankiai

6.1 Viena apdrojanti dalis

11 lentelė. Sparta su viena apdorojančia dalimi

Periodas 1, langa dydis 1					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,8 milijono				
Laikas (s)	11,146	11,267	11,426	11,078	
Sparta (vnt/s)	71774	71003	70015	72215	71190
AD	1				

Periodas 2, langa dydis 10					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,8 milijonai				
Laikas (s)	11,119	11,082	11,069	11,138	
Sparta (vnt/s)	71948	72189	72273	71826	72005
AD	1				

Periodas 5, langa dydis 50					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,848856 milijonai				
Laikas (s)	11,732	11,726	11,755	11,744	
Sparta (vnt/s)	72353	72390	72212	72279	72252
AD	1				

Periodas 10, langa dydis 100					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,857143 milijonai				
Laikas (s)	12,077	11,953	12,191	11,861	
Sparta (vnt/s)	70973	71709	70309	72265	71254
AD	1				

6.2 Dvi apdorojančios dalys

12 lentelė. Sparta su dviem apdorojančiomis dalimis

Periodas 1, langa dydis 1					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,857143 milijonai				
Laikas (s)	12,307	12,614	12,274	12,609	
Sparta (vnt/s)	69646	67951	69834	67978	68791
AD	2				

Periodas 2, langa dydis 10					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,834846 milijonai				
Laikas (s)	12,035	11,984	12,525	12,171	
Sparta (vnt/s)	69368	69663	66654	68593	68494
AD	2				

Periodas 5, langa dydis 50					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,833332 milijonai				
Laikas (s)	11,952	12,531	11,969	12,556	
Sparta (vnt/s)	69723	66501	69624	66369	67960
AD	2				

Periodas 10, langa dydis 100					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,829522 milijonai				
Laikas (s)	11,857	11,863	11,916	11,858	
Sparta (vnt/s)	69960	69925	69614	69954	69808
AD	2				

6.3 Keturios apdorojančios dalys

13 lentelė. Sparta su keturiomis apdorojančiomis dalimis

Periodas 1, langa dydis 1					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,8 milijono				
Laikas (s)	12,951	12,797	12,721	12,547	
Sparta (vnt/s)	61771	62514	62888	63760	62681
AD	4				

Periodas 2, langa dydis 10					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,8 milijono				
Laikas (s)	12,914	12,633	13,046	13,033	
Sparta (vnt/s)	61948	63326	61321	61382	61939
AD	4				

Periodas 5, langa dydis 50					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,75 milijono				
Laikas (s)	11,711	12,148	11,779	11,815	
Sparta (vnt/s)	64042	61738	63672	63478	63175

AD	4

Periodas 10, lango dydis 100					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,742119 milijono				
Laikas (s)	11,73	12,783	12,199	11,795	
Sparta (vnt/s)	63266	58055	60834	62918	61145
AD	4				

6.4 Aštuonios apdorojančios dalys

14 lentelė. Sparta su aštuoniomis apdorojančiomis dalimis

Periodas 1, lango dydis 1					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,545455 milijono				
Laikas (s)	11,488	12,003	11,436	11,485	
Sparta (vnt/s)	47480	45443	47696	47492	46975
AD	8				

Periodas 2, lango dydis 10					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,545455 milijono				
Laikas (s)	11,728	11,731	12,256	11,687	
Sparta (vnt/s)	46508	46496	44505	46671	45995
AD	8				

Periodas 5, lango dydis 50					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,545455 milijonai				
Laikas (s)	11,662	11,705	11,637	11,683	
Sparta (vnt/s)	46771	46600	46872	46687	46699
AD	8				

Periodas 10, lango dydis 100					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,571429 milijonai				
Laikas (s)	12,078	12,067	12,008	12,122	
Sparta (vnt/s)	47311	47354	47587	47139	47312
AD	8				

6.5 Šešiolika apdorojančių dalių

15 lentelė. Sparta su šešiolika apdorojančių dalių

Periodas 1, lango dydis 1					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,5 milijono				
Laikas (s)	13,691	14,349	13,59	13,473	
Sparta (vnt/s)	36520	34845	36791	37111	36269
AD	16				

Periodas 2, lango dydis 10					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,444444 milijono				
Laikas (s)	11,715	11,836	12,301	11,703	
Sparta (vnt/s)	37938	37550	36130	37976	37356
AD	16				

Periodas 5, lango dydis 50					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,444444 milijonai				
Laikas (s)	11,925	12,069	12,147	12,007	
Sparta (vnt/s)	37269	36825	36588	37015	36896
AD	16				

Periodas 10, lango dydis 100					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,48 milijonai				
Laikas (s)	13,134	13,028	13,12	12,934	
Sparta (vnt/s)	36546	36843	36585	37111	36742
AD	16				

6.6 Dvidešimt keturios apdorojančios dalys

16 lentelė. Sparta su dvidešimt keturiomis apdorojančiomis dalimis

Periodas 1, lango dydis 1					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,333333 milijono				
Laikas (s)	11,818	11,17	11,19	11,233	
Sparta (vnt/s)	28205	29841	29788	29674	29340
AD	24				

Periodas 2, lango dydis 10					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,324324 milijono				
Laikas (s)	10,643	10,654	10,662	10,748	
Sparta (vnt/s)	30472	30441	30418	30175	30355
AD	24				

Periodas 5, lango dydis 50					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,370296 milijonai				
Laikas (s)	11,984	11,903	11,92	12,017	
Sparta (vnt/s)	30899	31109	31065	30814	30945
AD	24				

Periodas 10, lango dydis 100					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,352941 milijonai				
Laikas (s)	11,772	11,717	11,721	11,875	

Sparta (vnt/s)	29981	30122	30111	29721	29962
AD	24				

7. Priedas. „Disruptor“ šablonas, Blocking

7.1 Viena apdorojanti dalis

17 lentelė. Sparta su viena apdorojančia dalimi

Periodas 1, langa dydis 1					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	40 milijonų				
Laikas (s)	12,258	10,604	10,526	11,599	
Sparta (vnt/s)	3263172	3772161	3800114	3448573	3556583
AD	1				

Periodas 2, langa dydis 10					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	20 milijonų				
Laikas (s)	11,677	11,654	12,193	11,647	
Sparta (vnt/s)	1712768	1716148	1640285	1717180	1695957
AD	1				

Periodas 5, langa dydis 50					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	30 milijonų				
Laikas (s)	12,31	12,261	12,359	12,294	
Sparta (vnt/s)	243704	244678	242738	244021	243783
AD	1				

Periodas 10, langa dydis 100					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	10 milijonų				
Laikas (s)	12,525	13,491	12,338	12,433	
Sparta (vnt/s)	79840	80057	81050	80431	80342
AD	1				

7.2 Dvi apdorojančios dalys

18 lentelė. Sparta su dviem apdorojančiomis dalimis

Periodas 1, langa dydis 1					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	60 milijonų				
Laikas (s)	15,995	20,05	15,708	19,86375	
Sparta (vnt/s)	3751172	2992518	3819709	2165908	3020577
AD	2				

Periodas 2, langa dydis 10

	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	19,202396 milijonų				
Laikas (s)	15,332	14,805	13,432	15,383	
Sparta (vnt/s)	1252439	1297021	1429600	1248286	1302917
AD	2				

Periodas 5, langa dydis 50					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	3 milijonai				
Laikas (s)	13,559	13,053	13,394	13,589	
Sparta (vnt/s)	221255	229832	223980	221092	223985
AD	2				

Periodas 10, langa dydis 100					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	1 milijonas				
Laikas (s)	12,845	12,876	13,095	12,9	
Sparta (vnt/s)	77851	77663	76365	77519	77345
AD	2				

7.3 Keturios apdorojančios dalys

19 lentelė. Sparta su keturiomis apdorojančiomis dalimis

Periodas 1, langa dydis 1					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	41,348133 milijonų				
Laikas (s)	25,099	24,431	11,119	26,884	
Sparta (vnt/s)	1647401	1692445	3718691	1538020	1889487
AD	4				

Periodas 2, langa dydis 10					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	14,644564 milijonų				
Laikas (s)	12,432	12,108	13,015	12,098	
Sparta (vnt/s)	1177973	1209494	1125204	1210494	1179752
AD	4				

Periodas 5, langa dydis 50					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	3 milijonai				
Laikas (s)	14,12	13,345	13,462	13,477	
Sparta (vnt/s)	212464	224803	222849	222601	220572
AD	4				

Periodas 10, langa dydis 100					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	1 milijonai				
Laikas (s)	13,587	13,714	13,155	13,11	
Sparta (vnt/s)	73599	72918	76016	76277	74674
AD	4				

7.4 Aštuonios apdorojančios dalys

20 lentelė. Sparta su aštuoniomis apdorojančiomis dalimis

Periodas 1, langa dydis 1					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	42,857143 milijonų				
Laikas (s)	16,608	26,185	11,705	11,28	
Sparta (vnt/s)	2580511	1636705	3661438	3799392	2606168
AD	8				

Periodas 2, langa dydis 10					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	9,988901 milijono				
Laikas (s)	22,033	23,289	21,695	22,124	
Sparta (vnt/s)	849396	833728	856167	827032	841418
AD	8				

Periodas 5, langa dydis 50					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	1,639512 milijonai				
Laikas (s)	11,638	11,974	11,086	11,099	
Sparta (vnt/s)	140875	136922	147890	147717	143198
AD	8				

Periodas 10, langa dydis 100					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,521739 milijonai				
Laikas (s)	10,385	10,962	10,322	10,548	
Sparta (vnt/s)	50239	47595	50546	49463	49434
AD	8				

7.5 Šešiolika apdorojančių dalių

21 lentelė. Sparta su šešiolika apdorojančių dalių

Periodas 1, langa dydis 1					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	9,316053 milijonų				
Laikas (s)	64,038	55,082	31,924	38,271	
Sparta (vnt/s)	145476	169130	291819	243423	196837
AD	16				

Periodas 2, langa dydis 10					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,584690 milijonų				
Laikas (s)	11,962	11,99	11,806	12,103	
Sparta (vnt/s)	585028	583661	592758	578212	584869
AD	16				

Periodas 5, langa dydis 50					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis

Duomenų kiekis	1,698915 milijonai				
Laikas (s)	12,268	12,539	12,178	12,337	
Sparta (vnt/s)	138483	135490	139506	137708	137781
AD	16				

Periodas 10, langa dydis 100					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,5 milijono				
Laikas (s)	10,667	10,709	10,549	10,66	
Sparta (vnt/s)	46873	46689	47397	46904	46964
AD	16				

7.6 Dvidešimt keturios apdorojančios dalys

22 lentelė. Sparta su dvidešimt keturiomis apdorojančiomis dalimis

Periodas 1, langa dydis 1					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,331217 milijono				
Laikas (s)	37,058	23,293	21,753	22,942	
Sparta (vnt/s)	8937	14219	15226	14437	12612
AD	24				

Periodas 2, langa dydis 10					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	6,129221 milijonų				
Laikas (s)	13,695	12,581	13,675	12,088	
Sparta (vnt/s)	447551	487180	448206	507050	471125
AD	24				

Periodas 5, langa dydis 50					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	1,414497 milijonai				
Laikas (s)	12,077	12,045	11,778	11,692	
Sparta (vnt/s)	117123	117434	120096	120979	118885
AD	24				

Periodas 10, langa dydis 100					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,514701 milijonai				
Laikas (s)	14,959	14,561	14,723	14,789	
Sparta (vnt/s)	42237	41298	41838	42254	41903
AD	24				

8. Priedas. „Disruptor“ šablonas, Busy wait

8.1 Viena apdorojanti dalis

23 lentelė. Sparta su viena apdorojančia dalimi

Periodas 1, langa dydis 1

	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	60 milijonų				
Laikas (s)	14,877	15,333	15,177	15,036	
Sparta (vnt/s)	4033071	3913128	3953350	3990422	3971997
AD	1				

Periodas 2, langas dydis 10					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	24 milijonai				
Laikas (s)	13,974	14,563	13,981	14,233	
Sparta (vnt/s)	1717475	1648012	1716615	1686222	1691600
AD	1				

Periodas 5, langas dydis 50					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	3 milijonai				
Laikas (s)	12,429	12,265	12,291	12,649	
Sparta (vnt/s)	241370	244598	244081	237172	241769
AD	1				

Periodas 10, langas dydis 100					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	1 milijonas				
Laikas (s)	12,434	13,675	12,385	12,392	
Sparta (vnt/s)	80424	73126	80742	80597	78607
AD	1				

8.2 Dvi apdorojančios dalys

24 lentelė. Sparta su dviem apdorojančiomis dalimis

Periodas 1, langas dydis 1					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	60 milijonų				
Laikas (s)	15,051	14,791	14,626	18,967	
Sparta (vnt/s)	3986446	4056520	4102283	3165057	3783996
AD	2				

Periodas 2, langas dydis 10					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	17,142857 milijonų				
Laikas (s)	13,269	13,374	11,069	12,573	
Sparta (vnt/s)	1291947	1281804	1548726	1363465	1363655
AD	2				

Periodas 5, langas dydis 50					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	3 milijonai				
Laikas (s)	12,712	13,381	13,226	12,958	
Sparta (vnt/s)	235997	224198	226825	231517	228546
AD	2				

Periodas 10, lango dydis 100					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	1 milijonas				
Laikas (s)	13,023	12,681	12,986	12,855	
Sparta (vnt/s)	76787	78858	77006	77790	77602
AD	2				

8.3 Keturios apdorojančios dalys

25 lentelė. Sparta su keturiomis apdorojančiomis dalimis

Periodas 1, lango dydis 1					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	60 milijonų				
Laikas (s)	15,38	14,573	14,937	15,062	
Sparta (vnt/s)	4901170	4117203	4016870	3983534	4004202
AD	4				

Periodas 2, lango dydis 10					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	15 milijonų				
Laikas (s)	12,428	12,86	13,219	14,864	
Sparta (vnt/s)	1206952	1166407	1134730	1009149	1124206
AD	4				

Periodas 5, lango dydis 50					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	3 milijonai				
Laikas (s)	13,595	13,426	13,432	13,694	
Sparta (vnt/s)	220669	223447	223347	219074	221618
AD	4				

Periodas 10, lango dydis 100					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	1 milijonas				
Laikas (s)	13,176	13,474	13,183	13,126	
Sparta (vnt/s)	75895	74217	75855	76184	75530
AD	4				

8.4 Aštuonios apdorojančios dalys

26 lentelė. Sparta su aštuoniomis apdorojančiomis dalimis

Periodas 1, lango dydis 1					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	40 milijonų				
Laikas (s)	13,604	13,614	13,534	14,208	
Sparta (vnt/s)	2940311	2938151	2955519	2815315	2911208
AD	8				

Periodas 2, lango dydis 10					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	12 milijonų				
Laikas (s)	15,009	15,526	15,61	15,664	
Sparta (vnt/s)	799520	772897	768737	766087	776585
AD	8				

Periodas 5, lango dydis 50					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	1,714286 milijono				
Laikas (s)	13,14	12,869	13,014	13,355	
Sparta (vnt/s)	130463	133210	131726	128362	130916
AD	8				

Periodas 10, lango dydis 100					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,545455 milijonas				
Laikas (s)	12,773	11,749	12,261	12,773	
Sparta (vnt/s)	42703	46425	44486	42703	44027
AD	8				

8.5 Šešiolika apdorojančių dalių

27 lentelė. Sparta su šešiolika apdorojančių dalių

Periodas 1, lango dydis 1					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	24,293957 milijonų				
Laikas (s)	11,427	14,809	13,816	14,002	
Sparta (vnt/s)	2126013	1640485	1758392	1735034	1797754
AD	16				

Periodas 2, lango dydis 10					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	12 milijonų				
Laikas (s)	20,141	19,092	20,07	19,06	
Sparta (vnt/s)	595799	628535	597907	629590	612533
AD	16				

Periodas 5, lango dydis 50					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	1,5 milijonai				
Laikas (s)	11,094	11,303	11,423	11,209	
Sparta (vnt/s)	135208	132708	131314	133821	133247
AD	16				

Periodas 10, lango dydis 100					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,545455 milijonas				
Laikas (s)	11,734	11,926	12,024	11,981	

Sparta (vnt/s)	46485	45736	45363	45526	45774
AD	16				

8.6 Dvidešimt keturios apdorojančios dalys

28 lentelė. Sparta su dvidešimt keturiomis apdorojančiomis dalimis

Periodas 1, langa dydis 1					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	19,609038 milijonų				
Laikas (s)	15,613	12,696	13,002	14,044	
Sparta (vnt/s)	1255942	1544505	1508155	1396257	1416965
AD	24				

Periodas 2, langa dydis 10					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	6,097483 milijonų				
Laikas (s)	12,069	14,471	12,902	14,283	
Sparta (vnt/s)	505218	421358	472599	426904	453977
AD	24				

Periodas 5, langa dydis 50					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	1,314082 milijonai				
Laikas (s)	11,048	11,84	11,953	11,508	
Sparta (vnt/s)	118942	110986	109937	114188	113407
AD	24				

Periodas 10, langa dydis 100					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,482541 milijonas				
Laikas (s)	11,396	11,403	11,592	11,381	
Sparta (vnt/s)	42343s	42317	41627	42398	42169
AD	24				

9. Priedas. Disruptor šablonas, Yelding

9.1 Viena apdorojanti dalis

29 lentelė. Sparta su viena apdorojančia dalimi

Periodas 1, langa dydis 1					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	46,875000 milijonai				
Laikas (s)	11,55	11,36	11,397	11,557	
Sparta (vnt/s)	4058441	4126320	4112924	4055983	4088173
AD	1				

Periodas 2, lango dydis 10					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	24 milijonų				
Laikas (s)	14,005	13,962	14,262	12,766	
Sparta (vnt/s)	1713673	1718951	1682793	1625355	1684358
AD	1				

Periodas 5, lango dydis 50					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	3 milijonai				
Laikas (s)	12,362	12,51	12,333	12,441	
Sparta (vnt/s)	242679	239808	243249	241138	241711
AD	1				

Periodas 10, lango dydis 100					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	1 milijonas				
Laikas (s)	12,463	12,481	12,456	12,356	
Sparta (vnt/s)	80237	80121	80282	80932	80392
AD	1				

9.2 Dvi apdorojančios dalys

30 lentelė. Sparta su dviem apdorojančiomis dalimis

Periodas 1, lango dydis 1					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	60 milijonų				
Laikas (s)	14,598	15,533	16,028	15,281	
Sparta (vnt/s)	4110152	3862743	3743448	3926444	3906250
AD	2				

Periodas 2, lango dydis 10					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	20 milijonų				
Laikas (s)	15,685	15,652	11,851	13,902	
Sparta (vnt/s)	1275103	1277791	1687621	1438641	1401296
AD	2				

Periodas 5, lango dydis 50					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	3 milijonai				
Laikas (s)	13,103	13,202	12,993	13,086	
Sparta (vnt/s)	228955	227238	230893	229252	229077
AD	2				

Periodas 10, lango dydis 100					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	1 milijonas				
Laikas (s)	12,651	12,937	12,961	12,904	
Sparta (vnt/s)	79045	77297	77154	77495	77740
AD	2				

9.3 keturios apdorojanti dalis

31 lentelė. Sparta su keturiomis apdorojančiomis dalimis

Periodas 1, lango dydis 1					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	60 milijonų				
Laikas (s)	14,65	14,839	14,518	14,786	
Sparta (vnt/s)	4095563	4043399	4132800	4057892	4082118
AD	1				

Periodas 2, lango dydis 10					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	15 milijonų				
Laikas (s)	15,857	15,266	12,307	12,641	
Sparta (vnt/s)	945954	982575	1218818	1186614	1070071
AD	1				

Periodas 5, lango dydis 50					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	3 milijonai				
Laikas (s)	13,491	13,464	13,539	13,959	
Sparta (vnt/s)	222370	222816	221582	214915	220373
AD	1				

Periodas 10, lango dydis 100					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,923077 milijonas				
Laikas (s)	12,095	12,165	12,166	12,873	
Sparta (vnt/s)	76318	75879	75873	77745	76446
AD	1				

9.4 Aštuonios apdorojančios dalys

32 lentelė. Sparta su aštuoniomis apdorojančiomis dalimis

Periodas 1, lango dydis 1					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	60 milijonų				
Laikas (s)	19,694	21,612	14,994	19,716	
Sparta (vnt/s)	3046613	2776235	4001600	3043213	3157230
AD	8				

Periodas 2, lango dydis 10					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	12 milijonų				
Laikas (s)	16,448	15,187	14,023	14,718	
Sparta (vnt/s)	729571	790149	855737	815328	795017
AD	8				

Periodas 5, lango dydis 50

	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	1,714286 milijono				
Laikas (s)	12,495	12,296	11,841	12,456	
Sparta (vnt/s)	137197	139418	144775	144775	139690
AD	8				

Periodas 10, langa dydis 100					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,545455 milijono				
Laikas (s)	11,804	10,959	11,382	11,273	
Sparta (vnt/s)	46209	49772	47922	48385	48038
AD	8				

9.5 Šešiolika apdorojančių dalių

33 lentelė. Sparta su šešiolika apdorojančių dalių

Periodas 1, langa dydis 1					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	28,715004 milijonai				
Laikas (s)	15,204	11,36	11,397	11,557	
Sparta (vnt/s)	1888647	2246343	2197352	1892756	2042827
AD	16				

Periodas 2, langa dydis 10					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	12 milijonų				
Laikas (s)	22,071	21,648	22,176	19,189	
Sparta (vnt/s)	543699	554323	541125	625358	564148
AD	16				

Periodas 5, langa dydis 50					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	1,714286 milijonai				
Laikas (s)	12,307	12,472	12,493	12,449	
Sparta (vnt/s)	139293	137450	137219	137153	137773
AD	16				

Periodas 10, langa dydis 100					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,5 milijonas				
Laikas (s)	10,852	10,798	10,732	10,973	
Sparta (vnt/s)	46074	46304	46589	45566	46130
AD	16				

Dvidešimt keturios apdorojančios dalys

34 lentelė. Sparta su dvidešimt keturiomis apdorojančiomis dalimis

Periodas 1, langa dydis 1					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis

Duomenų kiekis	17,142857 milijono				
Laikas (s)	10,259	12,586	11,0	12,611	
Sparta (vnt/s)	1671006	1362057	1558441	1359357	1476051
AD	24				

Periodas 2, lango dydis 10					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	6 milijonai				
Laikas (s)	12,194	11,379	13,336	11,512	
Sparta (vnt/s)	492045	527287	449910	521195	495652
AD	24				

Periodas 5, lango dydis 50					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	1,5 milijono				
Laikas (s)	12,933	12,456	12,818	12,379	
Sparta (vnt/s)	115982	120423	117022	121172	118609
AD	24				

Periodas 10, lango dydis 100					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,521739 milijono				
Laikas (s)	12,144	12,696	12,034	12,127	
Sparta (vnt/s)	42962	41094	43355	43022	42590
AD	24				

10. Priedas. Disruptor šablonas, sleeping

10.1 Viena apdorojanti dalis

35 lentelė. Sparta su viena apdorojančia dalimi

Periodas 1, lango dydis 1					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	60 milijonai				
Laikas (s)	14,899	15,116	18,212	14,812	
Sparta (vnt/s)	4027115	3969304	3294531	4050769	3807166
AD	1				

Periodas 2, lango dydis 10					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	20,609704 milijonų				
Laikas (s)	15,134	12,255	12,046	12,032	
Sparta (vnt/s)	1361814	1681738	1710916	1712907	1601780
AD	1				

Periodas 5, lango dydis 50					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	3 milijonai				
Laikas (s)	12,296	12,643	12,288	12,609	
Sparta (vnt/s)	243981	237285	244140	237925	240789
AD	1				

Periodas 10, langa dydis 100					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	1 milijonas				
Laikas (s)	12,464	12,369	12,415	12,512	
Sparta (vnt/s)	80231	80847	80547	79923	80385
AD	1				

10.2 Dvi apdorojančios dalys

36 lentelė. Sparta su dviem apdorojančiomis dalimis

Periodas 1, langa dydis 1					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	60 milijonų				
Laikas (s)	14,565	14,741	18,282	17,95	
Sparta (vnt/s)	4119464	4070280	3281916	3342618	3661997
AD	2				

Periodas 2, langa dydis 10					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	17,142857 milijonų				
Laikas (s)	12,253	12,715	10,817	12,466	
Sparta (vnt/s)	1399074	1348238	1584806	1375169	1421140
AD	2				

Periodas 5, langa dydis 50					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	3 milijonai				
Laikas (s)	13,309	13,197	13,484	13,591	
Sparta (vnt/s)	225411	227324	222485	220734	223959
AD	2				

Periodas 10, langa dydis 100					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	1 milijonas				
Laikas (s)	12,86	12,818	12,847	12,724	
Sparta (vnt/s)	77760	78015	77839	78591	78050
AD	2				

10.3 Keturios apdorojančios dalys

37 lentelė. Sparta su keturiomis apdorojančiomis dalimis

Periodas 1, langa dydis 1					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	60 milijonų				
Laikas (s)	14,772	15,061	14,841	14,775	
Sparta (vnt/s)	4061738	3983799	4042854	4060913	4037073
AD	4				

Periodas 2, langa dydis 10

	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	12 milijonų				
Laikas (s)	11,231	11,47	10,109	10,375	
Sparta (vnt/s)	1068471	1046207	1187061	1156626	1111497
AD	4				

Periodas 5, langa dydis 50					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	3 milijonai				
Laikas (s)	13,672	14,212	13,666	13,618	
Sparta (vnt/s)	219426	211089	219522	220296	217517
AD	4				

Periodas 10, langa dydis 100					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	1 milijonas				
Laikas (s)	13,158	13,01	13,609	13,193	
Sparta (vnt/s)	75999	76863	73480	75797	75514
AD	4				

10.4 Aštuonios apdorojančios dalys

38 lentelė. Sparta su aštuoniomis apdorojančiomis dalimis

Periodas 1, langa dydis 1					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	60 milijonų				
Laikas (s)	19,495	15,498	18,424	21,299	
Sparta (vnt/s)	3077712	3871467	3256621	2817033	3212163
AD	8				

Periodas 2, langa dydis 10					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	12 milijonų				
Laikas (s)	14,512	13,85	14,328	14,328	
Sparta (vnt/s)	826901	866425	837520	834666	841116
AD	8				

Periodas 5, langa dydis 50					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	3 milijonai				
Laikas (s)	11,654	12,466	11,72	12,398	
Sparta (vnt/s)	147098	137516	146270	138271	142152
AD	8				

Periodas 10, langa dydis 100					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,605144 milijonas				
Laikas (s)	13,707	12,159	11,946	12,165	
Sparta (vnt/s)	44148	49769	50656	49744	48433
AD	8				

10.5 Šešiolika apdorojančių dalių

39 lentelė. Sparta su šešiolika apdorojančių dalių

Periodas 1, langa dydis 1					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	24 milijonai				
Laikas (s)	14,085	10,357	14,247	12,065	
Sparta (vnt/s)	1703940	2317273	1684565	1989225	1891476
AD	16				

Periodas 2, langa dydis 10					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	7,448790 milijono				
Laikas (s)	11,853	12,113	11,932	12,565	
Sparta (vnt/s)	628430	614941	624270	592820	614802
AD	16				

Periodas 5, langa dydis 50					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	1,658604 milijonai				
Laikas (s)	11,783	11,694	11,739	11,824	
Sparta (vnt/s)	140762	141833	141290	140274	141037
AD	16				

Periodas 10, langa dydis 100					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,545455 milijonas				
Laikas (s)	11,265	11,509	11,462	11,321	
Sparta (vnt/s)	48420	47393	47588	48180	47892
AD	16				

10.6 Dvidešimt keturios apdorojančios dalys

40 lentelė. Sparta su dvidešimt keturiomis apdorojančiomis dalimis

Periodas 1, langa dydis 1					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	20,202020 milijono				
Laikas (s)	14,359	14,223	10,205	10,253	
Sparta (vnt/s)	1406923	1420376	1979619	1970352	1647799
AD	24				

Periodas 2, langa dydis 10					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	6,207433 milijonų				
Laikas (s)	12,235	12,167	12,716	12,072	
Sparta (vnt/s)	570350	510185	488159	514200	504771
AD	24				

Periodas 5, langa dydis 50					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis

Duomenų kiekis	1,437931 milijono				
Laikas (s)	12,165	11,702	11,974	12,629	
Sparta (vnt/s)	118202	122879	120087	113859	118665
AD	24				

Periodas 10, langо dydis 100					
	1 bandymas	2 bandymas	3 bandymas	4 bandymas	vidurkis
Duomenų kiekis	0,428571 milijonas				
Laikas (s)	10,19	10,419	10,105	10,022	
Sparta (vnt/s)	42057	41133	42411	42763	42082
AD	24				