https://doi.org/10.15388/vu.thesis.266
https://orcid.org/0000-0002-6956-4176

VILNIAUS UNIVERSITETAS

Albertas
JURGELEVIČIUS

Hibridinių paskirstytųjų skaičiavimų dalijimosi platforma

DAKTARO DISERTACIJA

Technologijos mokslai,
informatikos inžinerija (T 007)

VILNIUS 2021

Disertacija rengta 2016–2020 metais Vilniaus universitete.
Mokslinius tyrimus rėmė Lietuvos mokslo taryba.

Mokslinis vadovas:
prof. habil. dr. Leonidas Sakalauskas (Vilniaus universitetas, technologijos mokslai, informatikos inžinerija – T 007).

Mokslinis konsultantas:
dr. Virginijus Marcinkevičius (Vilniaus universitetas, technologijos mokslai, informatikos inžinerija – T 007).

https://doi.org/10.15388/vu.thesis.266
https://orcid.org/0000-0002-6956-4176

VILNIUS UNIVERSITY

Albertas
JURGELEVIČIUS

Hybrid public distributed computing platform

DOCTORAL DISSERTATION

Technological Sciences,
Informatics Engineering (T 007)

VILNIUS 2021

This dissertation was prepared in 2016 – 2020 at Vilnius University.
The research was supported by Research Council of Lithuania.

Academic supervisor:
Prof. Habil. Dr. Leonidas Sakalauskas (Vilnius University, Technological Sciences, Informatics Engineering – T 007).

Academic consultant:
Dr. Virginijus Marcinkevičius (Vilnius University, Technological Sciences, Informatics Engineering – T 007).

SANTRAUKA

Daiktų interneto ir išmaniąsias technologijas diegiant versle reikšmingai didėja skaičiuojamųjų išteklių poreikis. Atliekant skaičiavimus šiuolaikinėse verslo įmonėse paprastai apsiribojama turimų serverių ištekliais arba yra perkamos debesų kompiuterijos paslaugos. Tačiau tyrimai rodo, kad pačių įmonių turimi pajėgumai, naudojami vidiniams įmonės skaičiavimams, būna apkrauti tik nedidele dalimi. Neefektyvų informacinių technologijų panaudojimą lemia tai, kad dar nėra įmonėms sukurtų paskirstytųjų skaičiavimų platformų, kurios leistų panaudoti visus turimus kompiuterinius išteklius. Kuriant tokias platformas reikia išspręsti skirtingo našumo kompiuterių suderinamumo, aplinkos heterogeniškumo ir duomenų saugumo problemas. Siekiant išspręsti šias problemas reikia nagrinėti paskirstytųjų skaičiavimų platformas, dažniausiai pasitaikančias jų architektūras ir galimybes.

Darbo tikslas – sukurti hibridinių paskirstytųjų skaičiavimų platformą, veikiančią heterogeniškuose skaičiavimo tinkluose.

Šioje disertacijoje pasiūlyta hibridinių paskirstytųjų skaičiavimų platformos architektūra naudoja sukurtą užduočių paskirstymo tarp klasterių metodą. Nors šis užduočių paskirstymo metodas nėra naujas, jis dar nebuvo pritaikytas hibridinių paskirstytųjų skaičiavimo platformose. Pasirinktas metodas naudoja užduočių sulaikymo buferį, taip išvengdamas užduočių strigimo mažo našumo kompiuteriuose problemos. Nors šis metodas nereikalauja išankstinių duomenų apie užduočių vykdymo eilę ar joje esančias užduotis, jis gali veikti tik du klasterius turinčiose sistemose. Todėl sukurtas patobulintas užduočių paskirstymo metodas. Naujasis metodas veikia daugiau kaip du klasterius turinčiose sistemose ir prisitaiko prie paskirstytųjų skaičiavimų tinklo našumo pokyčių. Atlikta pasiūlytų algoritmų lyginamoji analizė su klasikiniu algoritmu, galinčiu veikti heterogeniškos sistemos sąlygomis, kai nėra informacijos apie vykdymui pateiktas užduotis.

Disertaciją sudaro įvadas, 4 skyriai, išvados, literatūros sąrašas. Disertacijos apimtis: 116 puslapių, 12 lentelių, 43 iliustracijos. Literatūros sąrašą sudaro 114 literatūros šaltinių. Tyrimų rezultatai publikuoti 2 recenzuojamuose periodiniuose mokslo žurnaluose, 1 tarptautinės konferencijos darbų CEUR leidinio straipsnyje, pristatyti tarptautinėse ir respublikinėse konferencijose.

Raktiniai žodžiai: paskirstytieji skaičiavimai, užduočių strigimo problema, užduočių paskirstymo algoritmai, paskirstytųjų skaičiavimų platformos.

ABSTRACT

The demand for computing resources in businesses companies is significantly increasing since the introduction of the Internet of Things and smart technologies. Computing in modern businesses is usually limited to the resources of available servers or cloud computing services. However, research shows that a significant part of their internal computing resource capacity is only lightly loaded. Since there are no distributed computing platforms designed for companies to use all available computing resources, this causes the inefficient use of their information technologies. Computer performance compatibility, environmental heterogeneity, and data security issues must be addressed to develop such a platform. It is necessary to examine distributed computing platforms, their most common architectures and capabilities to solve these problems.

The aim of this work is to create a hybrid distributed computing platform for heterogeneous computing networks.

The hybrid distributed computing platform architecture proposed in this dissertation uses the created task distribution method for tasks distribution between clusters. Although this method is not new, it has not yet been applied to hybrid distributed computing platforms. The proposed method uses a task stalling buffer, thus avoiding straggling tasks in slow-performing computers. Although this method does not require preliminary data on the task queue or its tasks, it can only work on systems with two clusters. Therefore, an improved task distribution method has been developed. The new method works in systems with more than two clusters and adapts to distributed computing environment performance changes. A comparative analysis of the proposed algorithms was performed with the classical algorithm capable of operating under heterogeneous system conditions when there is no preliminary data on the task queue or its tasks.

The dissertation consists of an introduction, 4 sections, conclusions, references. Dissertation consists of: 116 pages, 12 tables, 43 illustrations. The bibliography consists of 114 references. The research results have been published in 2 peer-reviewed scientific journals, 1 article of the international conference proceedings in the publication of CEUR, presented at international and national conferences.

Keywords: distributed computing, straggling task problem, task distribution algorithms, distributed computing platforms.

PADĖKA

Dėkoju gerbiamam darbo vadovui prof. habil. dr. Leonidui Sakalauskui už visą suteiktą pagalbą, palaikymą, laiku pateiktas įžvalgas, mokslinius patarimus ir vadovavimą. Nuoširdžiai dėkoju darbo konsultantui dr. Virginijui Marcinkevičiui už visapusę pagalbą atliekant mokslinius tyrimus, rengiant disertaciją, už visas konsultacijas ir pastabas. Labai dėkoju darbo recenzentams prof. habil. dr. Gintautui Dzemydai ir doc. dr. Sauliui Masteikai už tikslias įžvalgas, pastabas ir vertingus patarimus rengiant šį darbą.

Taip pat labai noriu padėkoti šeimos nariams ir artimiesiems už jų kantrybę, suteiktą motyvaciją ir nuolatinį palaikymą.

Albertas Jurgelevičius

Žymenys ir santrumpos

API	Taikomosios programos programavimo sąsaja (angl. *application programming interface*)
BOINC	Viešųjų paskirstytųjų skaičiavimo platforma (angl. *Berkeley open infrastructure for network computing*)
d	Intervalas sekundėmis tarp gaunamų naujų užduočių
D	Užduočių patekimo į sistemą intervalų rinkinys
DTS (D)	Eksperimento scenarijus, kuriame į sistemą patenkančių naujų užduočių srauto intensyvumas yra kintantis
EDF	Užduočių paskirstymo metodas (angl. *earliest deadline first*). Užduotys apdorojamos jų skubumo tvarka
FIFO	Elementų saugojimo duomenų struktūroje metodas (angl. *first in first out*). Pirmasis į struktūrą įtrauktas elementas ir paimamas pats pirmas, o paskutinis įtrauktas – paskutinis. FIFO mechanizmą naudoja eilė ir panašios duomenų struktūros
GUI	Grafinė vartotojo sąsaja (angl. *graphical user interface*)
i	Iteracijų skaičius vienai užduočiai atlikti
I	Galimų užduočių dydžių rinkinys
IT	Informacinės technologijos (angl. *information technology*)
k	Užduočių sulaikymo buferio ilgis
MVĮ	Mažos ir vidutinės įmonės (iki 50 darbuotojų)
PSO	Užduočių paskirstymo algoritmas (angl. *particle swarm optimisation*), iš galimų užduočių paskirstymo scenarijų kombinacijų parenkantis tinkamiausią
RAM	Darbinė kompiuterio atmintis (angl. *random-acess memory*)

STS (d)	Eksperimento scenarijus, kuriame į sistemą patenkančių naujų užduočių srauto intensyvumas yra nekintantis
TD (I)	Eksperimento scenarijus, kuriame vykdomos kintančio dydžio (skirtingo iteracijų kiekio) užduotys
TD_DTS	Eksperimento scenarijus, kuriame vykdomos kintančio dydžio užduotys ir yra kintantis užduočių patekimo į sistemą srauto intensyvumas
TD_STS	Eksperimento scenarijus, kuriame vykdomos kintančio dydžio užduotys ir yra nekintantis užduočių patekimo į sistemą srauto intensyvumas
TS (i)	Eksperimento scenarijus, kuriame vykdomos nekintančio dydžio (vienodo iteracijų kiekio) užduotys
TS_DTS	Eksperimento scenarijus, kuriame vykdomos nekintančio dydžio užduotys ir yra kintantis užduočių patekimo į sistemą srauto intensyvumas
TS_STS	Eksperimento scenarijus, kuriame vykdomos nekintančio dydžio užduotys ir yra nekintantis užduočių patekimo į sistemą srauto intensyvumas
TSB-dynamic	Užduočių paskirstymo algoritmas, naudojantis dinaminio ilgio užduočių sulaikymo buferį
TSB-static	Užduočių paskirstymo algoritmas, naudojantis statinio ilgio užduočių sulaikymo buferį
VM	Virtualioji mašina (angl. *virtual machine*)

Sąvokų žodynėlis

Atviras kompiuterių tinklas	Kompiuterių tinklas, leidžiantis viešai prieinamą naujų narių registraciją.
Apache Chronos	Darbų planuoklis, atsakingas už grafikų ir priklausomybių pagrindu vykdomų darbų planavimą.
Apache Mesos	Klasterių išteklių valdymo platforma.
Aukšto skaičiavimų pajėgumo klasteris	Klasterių tipas, kurį sudaro didelio našumo serveriai, sujungti didelio pralaidumo tinklu.
Debesų kompiuterija	Skaičiavimo išteklių, duomenų įkėlimo ir kitų kompiuterijos paslaugų teikimas internetu.
Didelių duomenų tyryba	Skaičiavimo procesas, leidžiantis dideliuose duomenų rinkiniuose atrasti duomenų sąryšių.
Docker	Paslaugų rinkinį teikianti platforma (angl. *paltform as a service*). „Docker“ naudoja operacinės sistemos lygio virtualizaciją ir suteikia galimybę susieti programinę įrangą paketuose, vadinamuose konteineriais (lengvesni už virtualiąsias mašinas). „Docker“ leidžia paleisti programinę įrangą įvairiose kompiuterių architektūrose ir operacinėse sistemose nereikalaudamas jokių programos pakeitimų.
Hibridiniai paskirstytieji skaičiavimai	Paskirstytųjų skaičiavimų metodas skaičiavimams atlikti, naudojantis kelias skirtingas infrastruktūras (klasterius).

Kompiuterių klasteriai	Kartu kompiuterių tinkle veikiančių kompiuterinių (virtualiųjų arba fizinių) grupė. Paskirstytieji skaičiavimai skiriasi nuo klasterinių skaičiavimų tuo, kad kompiuteriai paskirstytųjų skaičiavimų aplinkoje paprastai nevykdo bendrų užduočių, o klasteriniai kompiuteriai daug labiau susieti tarpusavyje dėl bendrų užduočių vykdymo. Be to, paskirstytųjų skaičiavimų sistemos sudarytos iš geografiškai plačiai paplitusių kompiuterių.
Paskirstytieji skaičiavimai	Kompiuterinio apdorojimo metodas, kurio metu skirtingos programos dalys tuo pat metu veikia keliuose kompiuterių tinklu komunikuojančiuose kompiuteriuose.
Privatieji paskirstytieji skaičiavimai	Skaičiavimams naudojamas uždaras kompiuterių tinklas, prie kurio jungtis ir skaičiavimuose dalyvauti gali tik leidimą turintys kompiuteriai.
Savanoriškoji kompiuterija	Paskirstytųjų skaičiavimų metodas, skaičiavimams naudojantis atviru kompiuterių tinklu sujungtus asmeninius kompiuterius.
Skaičiavimas kompiuterių tinkle	Paskirstytųjų skaičiavimų metodas, naudojantis tinklu sujungtų (dažniausiai internetu) kompiuterių išteklius.
Užduočių planuoklis	Kompiuterinės sistemos komponentas, suteikiantis galimybę suplanuoti programų ar scenarijų vykdymą iš anksto nustatytu laiku.
Užduočių sulaikymo buferis	Kompiuterinės sistemos komponentas, naudojamas laikinai saugoti užduočių

	duomenis, kol jie bus perkelti tolimesniam apdorojimui.
Viešieji paskirstytieji skaičiavimai	Paskirstytųjų skaičiavimų metodas, skaičiavimams naudojantis atviru kompiuterių tinklu sujungtus skaičiavimo išteklius.
VirtualBox	„Oracle VM VirtualBox“ programa, skirta kurti ir tvarkyti virtualiąsias mašinas. „VirtualBox“ suteikia aparatūros lygio virtualizaciją ir turi daugiau saugumo valdiklių už „Docker“. Tačiau virtualiosios mašinos naudoja daugiau kompiuterio išteklių, o jų paleidimas užima daugiau laiko nei konteinerių.

TURINYS

ĮVADAS

Tyrimų sritis ir problemos aktualumas

Daug šiuolaikinių įmonių ir organizacijų yra suinteresuotos rinkti ir apdoroti kiek galima daugiau su verslo procesais susijusių duomenų, kuriais vėliau būtų galima remtis sprendžiant įvairias verslo problemas ir siekiant priimti geresnius verslo sprendimus. Tam naudojami dirbtinio intelekto, mašininio mokymosi, statistikos ir kiti žinių gavimo metodai. Dėl vis didėjančio duomenų kiekio nebeįmanoma įgyvendinti tradicinių duomenų valdymo ir duomenų analizės metodų, pasinaudojant įprastiniais didelio našumo skaičiavimų ištekliais [90]. Verslo įmonėms ir organizacijoms kaupiant ir prisijungiant prie vis daugiau duomenų turinčių išteklių, skaičiavimo išteklių poreikis pradeda viršyti turimų vidinių IT infrastruktūrų galimybes. Tad organizacijos galiausiai nebeturi pakankamai vidinių skaičiavimo išteklių patenkinti paklausai. Turimi vidiniai aukšto skaičiavimų pajėgumo ištekliai ir panašūs tradiciniai duomenų valdymo sprendimai nebepajėgia susitvarkyti su tokiais duomenų kiekiais, o papildomo aukšto skaičiavimų pajėgumo klasterio diegimas ir priežiūra gali neatitikti finansinių galimybių [90]. Tokiais atvejais įmonės linkusios arba toliau didinti investicijas į savo vidinės IT infrastruktūros vystymą, arba ieškoti trečiųjų šalių sprendimų. Serverių pirkimas ar atnaujinimas yra brangus sprendimas, tad dažnai tenka ieškoti įmonių, kurios turi reikiamą infrastruktūrą ir siūlo savo sprendimus už prieinamą kainą.

Nors ir yra paskirstytųjų skaičiavimų sprendimų, leidžiančių lengvai sujungti vidinius IT išteklius į paskirstytųjų skaičiavimų platformą, organizacijos linkusios rinktis kitas alternatyvas. Vienas iš plačiai naudojamų sprendimų – debesų kompiuterija, kurios paslaugos nuolat plečiamos. Aukštus skaičiavimų pajėgumus siūlančios paslaugos tampa vis pigesnės, o šiuo metu nemažai išorinių kompanijų siūlo debesijos ir didžiųjų duomenų tyrybos sprendimus už prieinamą kainą. Norėdamos suvaldyti gaunamų duomenų srautus ir apdoroti turimus duomenis, įmonės dažnai naudojasi debesų kompiuterijos paslaugomis, kurias teikia tokios gerai žinomos įmonės kaip [97]:

- „Microsoft“[1];
- „Amazon“[2];

[1] http://azure.microsoft.com
[2] http://aws.amazon.com

- „Google“[3];
- „Rackspace“[4].

Tokiomis paslaugomis įmonės dažnai naudojasi tvarkydamos didžiulius turimų duomenų rinkinius. Iš duomenų gautos žinios gali padėti priimti teisingus verslo sprendimus ir suteikti klientams vertingos informacijos.

Tyrimai rodo, kad mažos ir vidutinės įmonės (MVĮ) neretai mano, jog trečiųjų šalių teikiamos debesų kompiuterijos paslaugos yra saugesnės už jų pačių kuriamus techninius sprendimus, naudojančius vidinę įmonės IT infrastruktūrą [65]. Tai verčia mažas ir vidutines įmones domėtis debesų kompiuterijos teikiamomis paslaugomis bei jų taikymo galimybėmis. Debesų kompiuterija nuolat plečiasi, nes ir toliau teikia našias skaičiavimo paslaugas už vis mažesnes kainas. Dėl šių priežasčių debesų kompiuterijos ir viešųjų paskirstytųjų skaičiavimų sprendimai yra aktualūs MVĮ.

Yra daugybė viešųjų paskirstytųjų skaičiavimų projektų, leidžiančių skirti savo turimus skaičiavimo išteklius, tačiau iki šiol nėra tinkamų programų, leidžiančių MVĮ panaudoti vidinius IT išteklius savo verslo poreikiams patenkinti – trūksta įrankių, kurie galėtų paversti vidinę IT infrastruktūrą (įskaitant darbuotojų asmeninius kompiuterius) patikima paskirstytųjų skaičiavimų platforma, atliekančia duomenų analizės ir įvairias kitas skaičiavimo išteklių reikalaujančias užduotis.

Viena iš pagrindinių problemų, neleidžiančių išnaudoti įmonių vidinių IT infrastruktūrų, yra užduočių strigimo hibridiniuose skaičiavimų tinkluose problema. Programos gali būti vykdomos paskirstytųjų skaičiavimų sistemose, tokiose kaip duomenų centrai ir klasteriai, naudojant išteklių valdytoją (YARN, „Apache Mesos“, „Borg“, BOINC ir kt.). Vykdomąją programą sudaro kelios mažesnės užduotys, apibrėžtos kaip mažiausi skaičiavimo vienetai, kurių vykdymą prižiūri išteklių valdytojas [41]. Tokios programos ir užduotys yra lygiagrečiai paskirstomos skirtingiems skaičiavimo ištekliams, siekiant paspartinti darbų vykdymą [63]. Tačiau tokiu būdu vykdant užduotis hibridiniuose skaičiavimo tinkluose pasitaiko užduočių strigimo problemų [43]. Neįprastai lėtai atliekama užduotis, lyginant su vidutine užduoties atlikimo trukme, vadinama įstrigusia [76]. Neįprastai lėta užduotis paprastai identifikuojama kaip bet kokia užduotis, kurios užduoties atlikimo laikas yra 50 % ilgesnis už vidutinį užduoties atlikimo laiką darbo etape [77, 83]. Lėtai vykdomos užduotys (angl. *stragglers*) daro įtaką viso darbo atlikimo ir užbaigimo laikui [5], didindamos išteklių naudojimą,

[3] https://cloud.google.com
[4] http://www.rackspace.com

mažindamos programų našumą [6, 39], sistemos prieinamumą ir didindamos papildomas eksploatacines išlaidas [42]. Atlikus didelio masto gamybos sistemų analizę [39] nustatyta, kad maždaug 4–6 % užduoties dalyvių neigiamai veikia daugiau kaip 50 % visų darbo vietų didesnėje sistemoje. Pastebėta, kad šis reiškinys taip pat pasireiškia duomenų centruose ir daro neigiamą įtaką programų veikimui. Užduočių strigimo problemų pasitaiko bet kurioje lygiagrečius skaičiavimus atliekančioje sistemoje, o dar labiau jos išryškėja vykdant darbus, kurie susideda iš daugybės užduočių ir yra atliekami daugelyje kompiuterių tuo pat metu.

Tai lėmė padidėjusį tyrimų skaičių, susijusių su pagrindinių užduočių strigimo priežasčių analize [30, 106], užduočių strigimo prognozavimu [79, 82] ir užduočių strigimo vengimo metodais [4, 5, 53, 111], įskaitant simuliacijų vykdymą [107], replikaciją, apkrovos balansavimą ir užduočių tvarkaraščio planavimą [24]. Kiekviename iš šių darbų daugiausia dėmesio skiriama tik tam tikro pogrupio užduočių ir sistemų taikymui.

Kitų autorių darbuose sprendžiant užduočių strigimo problemą paskirstytųjų skaičiavimų kompiuterinėse sistemose bandyta veiksmingai sumažinti neigiamą strigusių užduočių poveikį. Ši problema spręsta kuriant įvairius įstrigusių užduočių valdymo metodus.

Įstrigusių užduočių valdymo metodus galima suskirstyti į dvi pagrindines klases: aptikimas ir vengimas [45, 85]. Įstrigusių užduočių aptikimas apima metodus, leidžiančius atpažinti įstrigusių užduočių atsiradimą prieš arba po užduoties vykdymo skaičiavimų tinkle, pvz., atliekant įvykdytų užduočių analizę [52, 62] arba įstrigusių užduočių aptikimą naudojant „NearestFit“ [26]. Užduočių strigimo vengimo metodais daugiausia dėmesio skiriama bandymui išvengti [99] arba toleruoti aptiktą įstrigusią užduotį ir minimizuoti neigiamas pasekmes, pavyzdžiui, vykdyti užduočių ar išteklių prieigos planavimą, apkrovos balansavimą ir replikavimą [78, 80, 83]. Užduočių strigimo vengimo metodų pavyzdžiai yra „Dolly“ [4], GRASS [5], LATE [111] ir „Wrangler“ [53].

Darbo tikslas ir uždaviniai

Tyrimo tikslas: sukurti hibridinių paskirstytųjų skaičiavimų platformą, veikiančią heterogeniškuose skaičiavimo tinkluose.
Uždaviniai:

- įvertinti viešųjų paskirstytųjų skaičiavimų modelio taikymo poreikį ir galimybes;
- įvertinti esamas viešųjų paskirstytųjų skaičiavimų modeliu grįstas platformas;

- įvertinti esamus užduočių paskirstymo algoritmus, skirtus hibridinių paskirstytųjų skaičiavimų platformoms;
- sudaryti paskirstytųjų skaičiavimų platformos architektūrą ir eksperimentiniu būdu įvertinti platformos efektyvumą.

Tyrimo objektas ir metodai

Disertacijos tyrimo objektai:

- paskirstytieji skaičiavimai;
- užduočių įstrigimo problema;
- užduočių paskirstymo algoritmai.

Pagrindiniai tyrimo metodai taikomi disertacijoje – analitinė apžvalga, skaičiuojamieji eksperimentai, statistinė duomenų analizė. Tyrime buvo naudojami sugeneruoti duomenų rinkiniai, atliekamas platformos imitavimas, stebimos kompiuterių energetinės ir skaičiuojamosios apkrovos naudojant sistemų stebėjimo ir kompiuterinio tyrimo metodus.

Darbo naujumas

Šio darbo naujumą ir aktualumą sudaro tai, kad:

1. sistemiškai išnagrinėtas užduočių paskirstymo algoritmų taikymas hibridinėse heterogeniškose paskirstytųjų skaičiavimų aplinkose užduotims atlikti;
2. sukurta nauja hibridinių paskirstytųjų skaičiavimų platformos architektūra;
3. pritaikytas užduočių sulaikymo buferis užduotims paskirstyti;
4. sukurtas metodas, leidžiantis užduočių sulaikymo buferį naudoti paskirstytųjų skaičiavimų sistemose, turinčiose daugiau kaip du klasterius.

Darbo rezultatų praktinė reikšmė

Šio darbo rezultatų praktinę reikšmę sudaro tai, kad:

1. pritaikytas užduočių sulaikymo buferis, skirtas užduočių įstrigimo problemai spręsti hibridinių paskirstytųjų skaičiavimų sistemose;
2. sukurtas užduočių paskirstymo algoritmas, integruotas į hibridinių paskirstytųjų skaičiavimų klasterį. Jis leidžia sumažinti suminę užduočių vykdymo trukmę. Remiantis šiuo metodu parengta platforma, sujungianti viešąjį ir privatųjį klasterį į hibridinį skaičiavimų tinklą;

3. sukurta hibridinių paskirstytųjų skaičiavimų platforma, veikianti heterogeniškuose skaičiavimo tinkluose.

Darbo rezultatų aprobavimas

Disertacijos tyrimų rezultatai publikuoti 3 moksliniuose straipsniuose:

1. Jurgelevičius, A., Sakalauskas, L. BOINC from the View Point of Cloud Computing, CEUR Workshop Proceedings, 1973, 2017, 61–66;
2. Jurgelevičius, A., Sakalauskas, L. Big data mining using public distributed computing, Information technology and control. ISSN 1392-124X, eISSN 2335-884X, 2018, vol. 47(2), p. 236–248, DOI: 10.5755/j01.itc.47.2.19738;
3. Jurgelevičius, A., Sakalauskas, L., Marcinkevičius, V. Task stalling for a batch of task makespan minimisation in heterogeneous multigrid computing. Computational Science and Techniques, 2021, 8, 631–638, DOI: 10.15181/csat.v8.2103.

Autorius dalyvavo ir pristatė rezultatus 2-ose respublikinėse ir 4-iose tarptautinėse mokslinėse konferencijose:

1. respublikinėje konferencijoje „Informacinių technologijų iššūkiai kūrybos ekonomikoje“ (2017, Lietuva). Pranešimo tema: „BOINC karkaso taikymas didelių duomenų tyrybai“;
2. tarptautinėje konferencijoje „BOINC: Fundamental and Applied Science and Technology (BOINC:FAST 2017)“ (2017, Petrozavodskas, Rusija). Pranešimo tema: „BOINC from the view point of Cloud computing“;
3. XVIII tarptautinėje mokslinėje kompiuterininkų konferencijoje „Kompiuterininkų dienos – 2017“ (2017, Kaunas). Pranešimo tema: „Big Data mining using public distributed computing“;
4. respublikinėje konferencijoje „Duomenų analizės metodai programų sistemoms“ (2017, Druskininkai). Stendinio pranešimo tema: „BOINC Based Enterprise Desktop GRID“;
5. septintojoje tarptautinėje konferencijoje „Open International Conference on Electrical, Electronic and Information Sciences“ (2020, Lietuva). Skaityto pranešimo tema: „Distributed Hybrid Cloud Controller for Runbook Operations“;
6. ketvirtojoje tarptautinėje konferencijoje „4th International Conference on Innovations and Creativity“ (2020, Latvija). Skaityto pranešimo tema: „Task Stalling Buffer Application in Grid Computing“.

Ginamieji teiginiai

Disertacijos ginamieji teiginiai:

1. reikalingas naujas užduočių paskirstymo algoritmas, kuris užduočių įstrigimo heterogeniškuose paskirstytųjų skaičiavimų tinkluose problemą spręstų nenaudodamas užduočių replikacijos ir informacijos apie užduočių dydžius;
2. viešųjų paskirstytųjų skaičiavimų platformų skaičiavimo išteklių heterogeniškumo problema mažinama taikant pasiūlytą hibridinių paskirstytųjų skaičiavimų architektūrą, naudojančią užduočių sulaikymo buferį;
3. sukurtoje hibridinių paskirstytųjų skaičiavimų platformoje naudojamas užduočių paskirstymo algoritmas yra efektyvus užduočių paskirstymui daugiakanalėse paskirstytųjų skaičiavimų sistemose – sutrumpina užduočių įvykdymo laiką lyginant su FIFO algoritmu.

Disertacijos struktūra

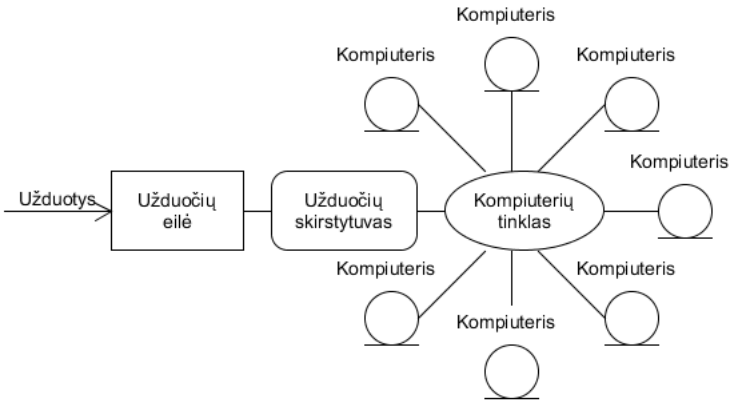
1 skyriuje pateikiama paskirstytųjų skaičiavimų modelio, esamų privačiųjų ir savanoriškųjų skaičiavimų sprendimų ir debesų kompiuterijos alternatyvų apžvalga. Be to, šiame skyriuje pateikiamos debesų kompiuterijos paslaugų problemos, nurodant alternatyvių sprendimų, tokių kaip viešųjų paskirstytųjų skaičiavimų, poreikį. 2 skyriuje apžvelgiamos esamos paskirstytųjų skaičiavimų platformos ir jų taikymai. Šiame skyriuje pateikiamos ir nagrinėjamos galimos BOINC platformos naudojimo galimybės siekiant pakeisti debesų kompiuterijos paslaugas. Galiausiai šiame skyriuje aprašomos paskirstytųjų skaičiavimų platformų keliamos problemos ir siūlomi galimi patobulinimai. 3 skyriuje pateikiami disertacijos darbo rezultatai: hibridinių paskirstytųjų skaičiavimų platforma, užduočių paskirstymo algoritmo modifikacija. Be to, apžvelgiami hierarchiniai ir nehierarchiniai užduočių paskirstymo algoritmai, kurie taip pat tinkami naudoti hibridinių paskirstytųjų skaičiavimų platformose, ir sprendžiama užduočių strigimo problema. 4 skyriuje pateikiami kompiuterinio modeliavimo ir eksperimentų rezultatai. Galiausiai pateikiamos galutinės disertacijos išvados.

Disertacijos apimtis: 116 puslapių, 12 lentelių, 43 iliustracija. Disertacijoje remtasi 114 literatūros šaltinių.

1. PASKIRSTYTŲJŲ SKAIČIAVIMŲ MODELIS

1.1 Paskirstytieji skaičiavimai

Paskirstytieji skaičiavimai – kompiuterinio apdorojimo metodas, kai skirtingos programos dalys tuo pat metu veikia keliuose kompiuteriuose, komunikuojančiuose per tinklą (žr. 1 pav.). Paskirstytieji skaičiavimai yra dalinio arba kitaip – lygiagretaus skaičiavimo tipas, tačiau pastarasis terminas labiau apibrėžia skaičiavimą, kai skirtingos programos dalys tuo pat metu veikia keliuose procesoriuose, kurie yra to paties kompiuterio dalis. Nors abu skaičiavimo tipai reikalauja, kad programa būtų padalyta į segmentus (dalis, kurios veikia tuo pačiu metu), paskirstytiesiems skaičiavimams dar reikia įvertinti skirtingas aplinkas, kuriose veikia atskiros programos dalys. Pavyzdžiui, atskiri kompiuteriai tikriausiai turės skirtingas failų sistemas ir skirtingus aparatinės įrangos komponentus.



1 pav. Paskirstytųjų skaičiavimų heterogeninės sistemos modelis

1.2 Viešieji paskirstytieji skaičiavimai

Viešieji paskirstytieji skaičiavimai yra skaičiavimo metodas, kai tuo pat metu naudojami keli viešieji kompiuteriai skaičiavimams atlikti. Komunikacija atliekama tinklu, naudojant kliento-serverio architektūrą, kur kliento mazgai pateikia savo turimus išteklius, kad gautų naujų užduočių iš serverio. Skaičiavimai įprastai atliekami lygiagrečiai, nedarant jokios įtakos kitiems skaičiavimo mazgams. Pagrindinis paskirstytųjų skaičiavimų sistemos tikslas – sujungti skaičiavimo išteklius į dinamišką, atvirą skaičiavimų tinklą. Tai efektyvus būdas panaudoti skaičiavimo išteklius, kurių potencialas kitu atveju nebūtų iki galo išnaudotas. Vienas iš šio modelio

pranašumų, lyginant su debesų kompiuterija, yra tas, kad viešiesiems paskirstytiesiems skaičiavimams reikalingą infrastruktūrą gali visiškai įrengti ir valdyti IT infrastruktūrą turinti organizacija. Taip gali būti sumažintos išlaidos, panaikinant poreikį įmonėms pirkti debesų kompiuterijos paslaugas. Be to, tai apsaugo nuo kai kurių problemų, kylančių siekiant apsaugoti duomenų privatumą, kai naudojamos debesų kompiuterijos paslaugos.

Berklio atviro tinklo skaičiavimo infrastruktūra (angl. *Berkeley open infrastructure for network computing*, BOINC) yra populiariausia [8, 44] atvirojo kodo tarpinės (programinė įranga, veikianti tarp operacinės sistemos ir vykdomosios programos, angl. *middleware*) programinės įrangos sistema, palaikanti savanoriškąją kompiuteriją (angl. *volunteer computing*) ir skaičiavimus klasteriuose (angl. *grid computing*). Tai yra geras tokio skaičiavimo metodo pavyzdys, kur didelių skaičiavimo išteklių reikalaujančios užduotys dalijamos į mažesnes užduotis, o šios paskui paskirstomos daugeliui kompiuterių. Po to rezultatai agreguojami ir gaunamas galutinis sprendžiamos užduoties atsakymas. Šiuo tikslu naudojama kliento-serverio architektūra, kur kliento mazgai iš serverio prašo užduočių ir su užduotimis susijusių duomenų. Tada skaičiavimo rezultatai siunčiami atgal į serverį. Skaičiavimai vykdomi be vidinių procesų komunikacijos ir yra naudojama tik bendra duomenų bazė. Vieni iš populiariausių projektų yra šie:

1. CERN + KC Gigabito skaičiavimo iššūkis[5];
2. „Gridcoin“[6];
3. „SETI@home“[7].

Toliau apžvelgiamas viešųjų paskirstytųjų skaičiavimų metodo taikymas savanoriškoje kompiuterijoje.

1.3 Savanoriškoji kompiuterija

Savanoriškoji kompiuterija naudoja viešųjų paskirstytųjų skaičiavimų metodą. Šis metodas naudojamas spręsti užduotims tinklu sujungtuose viešuosiuose kompiuteriuose. Metodas naudoja kliento-serverio modelį, pagal kurį kliento mazgai teikia savo turimus laisvus išteklius projektą valdančiam serveriui. Tinklo ryšys reikalingas, kad mazgai galėtų susisiekti su serveriu. Tai leidžia klientams paprašyti serverio atsiųsti naujas užduotis ir serveriui atgal grąžinti rezultatus. Kai kuriais atvejais mazgai gali būti nustatomi

[5] https://cernkcchallenge.github.io/CernKCChallenge/
[6] http://gridcoin.us
[7] http://setiathome.ssl.berkeley.edu

bendrauti tarpusavyje, tačiau užduotys paprastai būna individualios ir vykdomos tuo pat metu.

Paskirstytiesiems skaičiavimams gali būti naudojamas viešųjų skaičiavimų išteklių pritraukimas. Šis modelis vadinamas savanoriškąja kompiuterija. Šis modelis priklauso nuo to, ar žmonės ir organizacijos skiria procesoriaus laiko, tinklo ir saugojimo galimybių iš jiems priklausančių kompiuterių. Taip skaičiavimo išteklius galima sujungti į atvirą dinaminį tinklą, kuriame galima lengvai pridėti naujus mazgus, o senus pašalinti. Paskirstytieji skaičiavimai – efektyvus būdas panaudoti skaičiavimo mazgų išteklius, kurie kitu atveju būtų švaistomi. Tokia infrastruktūra nereikalauja jokių papildomų išlaidų ir netgi gali sumažinti jau esamas, todėl šis modelis turi daug privalumų. Be to, uždarame tinkle galima sukurti paskirstytųjų skaičiavimų modeliu grįstą infrastruktūrą. Taip infrastruktūra bus apribota mazgais, kurie yra laikomi organizacijos patalpose. Prireikus skaičiavimų tinklą gali visiškai kontroliuoti ir vystyti įmonės darbuotojai. Lyginant su tokiomis populiariomis alternatyvomis kaip debesų kompiuterija, šis modelis turi du pagrindinius pranašumus:

- sumažinamos vidinės IT infrastruktūros ir išorinių paslaugų pirkimo išlaidos;
- sumažinamos duomenų privatumo ir saugumo problemos.

Prie tinklo prijungti savanoriškosios kompiuterijos ištekliai teikia išteklius didelių skaičiavimų reikalaujančioms užduotims spręsti. Šie ištekliai apima procesoriaus, GPU, RAM, atminties ir interneto ryšio išteklius. Jie gaunami iš tuo metu neužimtų kompiuterių, naudojant centralizuotą serverio-kliento modelį. Vienas iš svarbiausių savanoriškąja kompiuterija paremtų sistemų aspektų, į kurį reikia atsižvelgti, yra tas, kad būtina pritraukti ir įtikinti savanorius dalyvauti. Populiariausias būdas pritraukti savanorius – juos apdovanoti taškais už dalyvavimą. Taškai įvertinami apskaičiuojant savanorio indėlį į mokslinę projekto pažangą. Tačiau šie taškai paprastai yra tik matavimo priemonė ir savanoriams nesuteikia jokios turtinės vertės. Mainais suteikti ištekliai priskiriami prie įvairių viešųjų projektų, vykdant su projektais susijusias individualias užduotis. Daugelyje tokių projektų naudojama BOINC sistema, kuri apžvelgta 2 skyriuje [11, 12, 28].

Savanorių arba minios (angl. *crowd*) kompiuterija yra populiarus metodas sprendžiant sudėtingas įvairių sričių tyrimų problemas [73]. Ji leidžia naudotojui pamiršti apie tam tikras išlaidas, susijusias su fizinės infrastruktūros pirkimu ir priežiūra, ir padeda viešinti projektą visuomenei [44]. Sukurti savo specializuotą duomenų tvarkymo centrą gali būti per brangu arba netinkama. Tačiau naudojant šį modelį gali kilti problemų, jei duomenų

analizės poreikis yra nereguliarus [55]. Savanorių skaičiavimai susideda iš dviejų aspektų: skaičiavimo ir dalyvavimo [12]. Šie aspektai yra susiję su šių problemų sprendimu:

- didelių skaičiavimo išteklių reikalaujančių darbų paskirstymas ir valdymas. Dauguma skaičiavimo veiklos rūšių, kurios priklauso nuo naudotojo interaktyvios įvesties, visiškai neapkrauna kompiuterių. Dėl to didelė dalis išteklių nepanaudojama, o šie ištekliai gali būti panaudoti papildomiems skaičiavimams [10];
- asmenų skatinimas ir įtikinimas paaukoti savo skaičiavimo išteklius projektui. Turi būti įdiegtas patikimas saugumo mechanizmas, užtikrinantis, kad naudotojai bus apsaugoti nuo galimų saugumo problemų [11]. Pasitikėjimas motyvuoja dalyvius ne tik dalytis turimais ištekliais, bet ir suteikti daugiau prieigos teisių prie savo kompiuterių [68].

1.3.1 Vikinomika

Verslo aplinkoje masinis bendradarbiavimas gali būti vertinamas kaip verslo procesų perleidimo trečiosioms šalims išplėtimas, perduodant buvusias vidaus verslo funkcijas kitiems verslo subjektams. Šis procesas yra dar vadinamas Vikinomika (angl. *Wikinomics*). Vikinomika apibūdina plataus bendradarbiavimo ir naudotojų dalyvavimo padarinius bei tai, kaip keičiasi santykiai tarp verslo ir rinkų. Skirtumas tarp tradicinių verslo procesų ir masinio bendradarbiavimo yra tas, kad užuot turint organizuotą verslo organą, specialiai sukurtą unikaliai funkcijai atlikti, masinis bendradarbiavimas priklauso nuo atskirų laisvų agentų, suburtų bendradarbiauti ir patobulinti tam tikrą operaciją ar išspręsti problemą. Šis skirtumas atspindimas ir kaip išorinių išteklių telkimas, kurį galima paskatinti naudojant atlygio sistemą (tačiau tai nėra būtina).

Šie keturi principai yra pagrindinės įmonės vikinomikos sąvokos:

- atvirumas, apimantis ne tik atvirus standartus ir turinį, bet ir finansinį skaidrumą bei atvirą požiūrį į išorines idėjas ir išteklius;
- bendradarbiavimas, kuris pakeičia hierarchinius modelius bendradarbiavimui tinkamesniu forumu;
- dalijimasis, skatinantis atvirą požiūrį į produktus, intelektinę nuosavybę, mokslo žinias;
- veiksmų globalumas, kuris apima globalizaciją ir fizinių bei geografinių ribų nepaisymą tiek įmonės, tiek individualiu lygmeniu.

Vikinomika parodo, kad masinis bendradarbiavimas ir savanoriškoji kompiuterija verslo aplinkoje nėra naujos sąvokos. Šie metodai taikomi versle, egzistuoja tam skirtos platformos, kurios apžvelgtos tolimesniuose skyriuose.

1.3.2 Savanoriškosios kompiuterijos modelio išplėtimas

Yra daug atliktų mokslinių tyrimų, tiriančių, kaip būtų galima apskaičiuoti ir gauti reikiamą išteklių kiekį norimiems skaičiavimams atlikti. Vienas iš jų pristatytas „EU FP7 EDGI“ projekte ir apžvelgtas [68]. Projektas sujungia BOINC ir „XtreemWeb“ darbinių kompiuterių klasterius su debesų kompiuterijos paslaugomis. Tai praplečia šiuos klasterius naujais ištekliais pagal pareikalavimą (angl. *on-demand*), todėl šis sprendimas tampa panašus į programinės įrangos kaip paslaugos sprendimą (angl. *software as a service,* SaaS) [68]. Naudotojams nereikia patirti išlaidų dėl papildomų išteklių, nes jie gaunami iš savanorių. Be to, šis sprendimas pagerina reagavimo laiką savanoriškosios kompiuterijos modelį naudojančiose sistemose. Savanorių debesija gali būti debesų kompiuterijos paradigmos patobulinimas, kur debesijos ištekliai būtų gaunami iš savanorių [68]. Tai – didelis privalumas, nes didžiųjų duomenų tyryba reikalauja didelių skaičiavimo išteklių, kurių daugelis įmonių ir organizacijų negali sau leisti patirti. [60] atliko tyrimą, kur savanorių darbinių kompiuterių klasterio išteklių poreikis papildytas ištekliais naudojant debesų kompiuteriją. Žinoma, nepastovumo ir prieinamumo problemos sukelia didelių sunkumų didžiųjų duomenų tyrybos platformose, kurios aptariamos 2 skyriuje.

„Clouds@home“ yra dar viena debesų kompiuterijos paslaugas teikianti sistema, paremta savanoriškąja kompiuterija. Ši sistema laikoma nauja debesų kompiuterijos forma. Projekto tikslas buvo sukurti mažos apimties ir pigią debesų kompiuterijos infrastruktūrą sujungiant debesų ir savanoriškosios kompiuterijos modelius. Iš savanoriškosios kompiuterijos ji sukuria panašią į debesų kompiuterijos paslaugas teikiančią infrastruktūrą. Idėja remiasi virtualizacijos technologijos panaudojimu savanorių skaičiavimo ištekliuose [33]; tai – metodas, vadinamas „programų smėliadėže“ (angl. *application sandboxing*). Ji izoliuoja programą virtualios mašinos (VM) viduje ir naudoja pagalbinę programą (angl. *wrapper*), kad paleistų VM ir tvarkytų joje veikiančias programas [67, 68].

Nepaisant pastangų tobulinti ir plėsti šią technologiją, savanoriškoji kompiuterija vis dar ne visada laikoma geriausiu visų problemų sprendimo būdu. Tačiau sprendimas nebūtinai turi būti pagrįstas savanoriškais skaičiavimais. Anot [55], kai kuriais atvejais gali būti sunku ar netinkama

bandyti pritraukti skaičiavimo išteklius iš išorės. Apdorojimui skirti duomenys gali būti konfidencialūs ir gali reikėti analizuoti didelius duomenų kiekius. Mat atliekant didžiųjų duomenų tyrybos užduotis galima lengvai įtraukti duomenų rinkinius, kuriuose yra privačių duomenų. Tai rodo, kad išteklių panaudojimo sprendimai neturėtų remtis tik savanoriškąja kompiuterija ir turėtų arba integruoti kitas platformas, arba taikyti kai kuriuos duomenų apsaugos metodus. Tai plačiau aptariama 2 skyriuje.

1.4 Privatieji paskirstytieji skaičiavimai

Privačiųjų paskirstytųjų skaičiavimų modelis naudoja kliento-serverio architektūrą ir yra nukreiptas į didelį vidinių išteklių panaudojimą ir našumą. Privatieji paskirstytieji skaičiavimai – tinkamiausias modelis įmonėse ir organizacijose, nes teikia aukštos kokybės paslaugas, našumą ir duomenų saugumą.

Šis modelis taip pat naudojamas debesų kompiuterijoje. Duomenų saugumas debesų kompiuterijoje yra sudėtingesnis nei duomenų saugumas tradicinėse informacinėse sistemose [97]. Tradicinių saugumo mechanizmų, tokie kaip tapatybės atpažinimas ir įgaliojimų suteikimas dabartine forma, nebepakanka užtikrinti saugumui debesų kompiuterijoje. Tyrimai rodo, kad debesų kompiuterija suteikia didesnį rizikos lygį, nes esminės paslaugos dažnai perkamos iš trečiųjų šalių. Tad tampa sudėtingiau užtikrinti duomenų saugumą ir privatumą, užtikrinti duomenų ir paslaugų prieinamumą ir patikrinti reikalavimų laikymąsi. Neskaitant to, kad debesų kompiuterijos įdiegimas suteikia daug naudos, yra ir tam tikrų reikšmingų kliūčių, kurias reikia pašalinti siekiant įdiegti sistemą. Viena iš reikšmingiausių kliūčių yra saugumas, po jo eina kliūtys, susijusios su reikalavimų laikymusi, privatumu ir teisiniais aspektais. Duomenų saugojimas, virtualizacija ir kompiuteriniai tinklai – didžiausi saugumo iššūkiai debesų kompiuterijoje [47]. Atsirandančios naujos technologijos, tokios kaip daiktų internetas (angl. *cloud of things*), sukurs naujų galimybių verslui, tačiau taip pat didins išpuolių grėsmę [1]. Būtina apsaugoti duomenis nuo bet kokios neteisėtos naudotojų prieigos ar išpuolių, tokių kaip tarnybinės prieigos uždraudimo (angl. *denial of service*), duomenų modifikavimo ar klastojimo [71]. Saugumas, privatumas ir tapatybės apsauga tampa labai svarbūs hibridinėje debesų kompiuterijoje, kurią dažniausiai naudoja verslas, sujungdamas privačiosios bei viešosios debesų kompiuterijos teikiamas paslaugas. Siekiant padaryti debesų kompiuteriją prieinamą diegti privatiems naudotojams ir įmonėms, pirmiausia turi būti išspręsti naudotojų saugumo klausimai. Tada debesų kompiuterijos aplinka tampa patikima. Tai yra pagrindinė ir būtinoji sąlyga siekiant laimėti

naudotojų pasitikėjimą bei įtikinti juos diegti šią technologiją [47, 71, 97]. Privatumo problemos skiriasi priklausomai nuo skirtingų debesijos paslaugų naudojimo scenarijų ir gali būtų paskirstytos taip [97]:

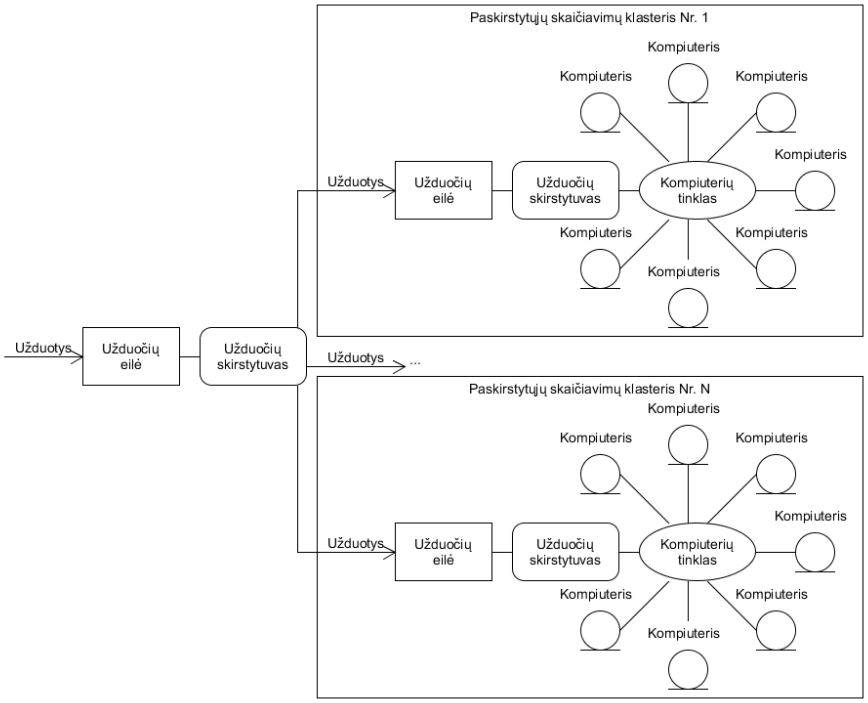
1. problema siekiant išvengti piktybinio duomenų panaudojimo ir neteisėto duomenų perpardavimo;
2. problema siekiant išvengti duomenų praradimo, nutekinimo, neteisėto modifikavimo ar padirbinėjimo replikuojant duomenis;
3. problema sprendžiant, kuri šalis atsakinga už teisinių asmens duomenų saugojimui ir apdorojimui keliamų reikalavimų užtikrinimą;
4. problema sprendžiant, kokiu mastu debesijos paslaugas teikiantys subrangovai yra įtraukiami apdorojant duomenis ir kokiu mastu jie gali būti identifikuojami, tikrinami ir patvirtinami.

1.5 Hibridiniai paskirstytieji skaičiavimai

Duomenys tapo vienu iš kritiškiausių ir vertinamiausių turtų šiuolaikiniame sparčiai kintančiame verslo pasaulyje. Organizacijos renka ir naudoja duomenis, kad įvertintų pagrindinius veiklos rodiklius, priimtų pagrįstus sprendimus ir nustatytų savo tikslus. Naudingi duomenys gali padėti rasti problemas, padidinti verslo efektyvumą, rasti naujų galimybių ir išlikti konkurentų priešakyje. Dėl vykstančių pramonės gamybos pertvarkymų ir skaitmeninimo („Pramonės 4.0 strateginė iniciatyva“[8]) duomenų kiekis yra linkęs didėti [86].

Didelės kompanijos paprastai sprendžia aparatūros pajėgumų problemas atnaujindamos esamus ar pirkdamos naujus serverius ir samdydamos papildomų darbuotojų sistemoms prižiūrėti. Mažos ir vidutinės įmonės paprastai neturi finansinių galimybių tokioms investicijoms. Dažnai mažesnės įmonės perka išorines debesų kompiuterijos paslaugas naudodamos įvairias paslaugų prenumeratas arba paslaugų pirkimo pagal pareikalavimą schemas. Šios paslaugos suteikia saugų, dinaminio dydžio atminties ir skaičiavimų pajėgumą. Tyrimai rodo, kad dėl to labiau prieinamas savanoriškosios kompiuterijos modelis atrodo nepatikimas ir pernelyg sunkiai pritaikomas [58].

[8] https://ec.europa.eu/growth/tools-databases/dem/monitor/sites/default/files/DTM_Policy%20initiative%20comparison%20v1.pdf



2 pav. Hibridinių paskirstytųjų skaičiavimų sistemos modelis

Savanoriškosios kompiuterijos modelis leidžia savanoriams teikti turimus skaičiavimo išteklius projektams. Nors šis modelis gali sumažinti paslaugų kaštus, jam taip pat trūksta patikimumo. Ne visada gali būti prieinamas reikiamas savanorių skaičiaus arba jie ne visada atliks paskirtas užduotis. Be to, privačių duomenų apsauga gali sukelti papildomų problemų. Asmens duomenų privatumo problemos yra ypač aktualios dabar, nes nuo 2018 m. gegužės 25 d. įmonės ir organizacijos privalo laikytis BDAR[9] (bendrojo duomenų apsaugos reglamento) taisyklių Europos Sąjungoje.

Todėl dabar susiduriame su hibridinių paskirstytųjų skaičiavimų (žr. 2 pav.) ir paskirstytosios debesijos koncepcija, kuri yra viena iš 10 geriausių „Gartner“[10] strateginių technologijų tendencijų 2020 bei 2021 m. Paskirstytoji debesija yra viešųjų debesijos paslaugų paskirstymas skirtingose geografinėse vietose. Nors tokios paslaugos yra ne fiziniuose duomenų centruose, jas vis tiek kontroliuoja ir prižiūri paslaugų teikėjas. Ši technologija, be privataus

[9] https://gdpr.eu

[10] https://www.gartner.com/smarterwithgartner/gartner-top-10-strategic-technology-trends-for-2020/

klasterio teikiamų paslaugų, suteikia dar ir viešosios debesų kompiuterijos paslaugų pranašumus.

Nepaisant pranašumų, paskirstytosios hibridinės debesų kompiuterijos modelis kelia įvairių iššūkių. Vienas iš tokių iššūkių – užduočių planavimas ir vykdymas. Būtina išlaikyti optimalų darbo krūvį tarp klasterių. Tačiau esami užduočių planavimo algoritmai negali subalansuoti darbo krūvio be jokios papildomos informacijos apie užduotis (pvz., užduoties dydį, kiekį ir gaunamų užduočių srautą). Duomenų ir sistemų saugumo užtikrinimas taip pat yra didelis iššūkis, nes duomenys platinami tarp skirtingų įrenginių, įskaitant serverius, asmeninius kompiuterius, įvairius mobiliuosius įrenginius, tokius kaip belaidžių jutiklių tinklai (angl. *wireless sensor networks*) ir išmanieji telefonai.

1.6 Skyriaus išvados

Apibendrinant pirmą skyrių, gautos šios išvados:

1. šaltinių apžvalga parodė, kad efektyviam didelių duomenų apdorojimui naudojant viešuosius paskirstytuosius skaičiavimus rekomenduojama taikyti BOINC platformą;
2. šaltinių apžvalga parodė, kad sprendžiant viešųjų paskirstytųjų skaičiavimų platformų skaičiavimo išteklių heterogeniškumo problemą reikia išnagrinėti hibridinių paskirstytųjų skaičiavimų architektūras ir jų taikymo galimybes.

2. PASKIRSTYTŲJŲ SKAIČIAVIMŲ PLATFORMOS IR JŲ TAIKYMAS

Šiame skyriuje nagrinėjami viešųjų paskirstytųjų skaičiavimų modelio pritaikymo verslo aplinkoje iššūkiai ir sprendimo būdai. Apžvelgiamos egzistuojančios platformos, taip pat nagrinėjami esami BOINC pagrindu veikiantys sprendimai, teikiantys alternatyvias debesų kompiuterijos paslaugas. Apžvelgiami šių sprendimų našumas ir paslaugų kokybė. Aptariami matematiniai metodai, padedantys padidinti paslaugų patikimumą ir pritaikantys BOINC bei panašius sprendimus, tinkamus verslui [57, 58].

2.1 Viešųjų paskirstytųjų skaičiavimų platformos

Viešųjų paskirstytųjų skaičiavimų modelis leidžia apjungti viešuosius kompiuterius lygiagrečiam paskirstytųjų užduočių vykdymui. Šiuo modeliu siekiama išspręsti heterogeninės aplinkos problemas, leidžiant naujiems išoriniams skaičiavimo mazgams prisijungti prie skaičiavimų. Jis naudoja kliento-serverio architektūrą, kuri įgalina mazgus teikti išteklius projekto serveriui. Šis modelis leidžia skaičiavimo mazgams paprašyti pagrindinio serverio duoti naujų užduočių ir grąžinti rezultatus. Viešųjų paskirstytųjų skaičiavimų metodai savo skaičiavimo pajėgumais gali konkuruoti su esamais debesų kompiuterijos sprendimais [70].

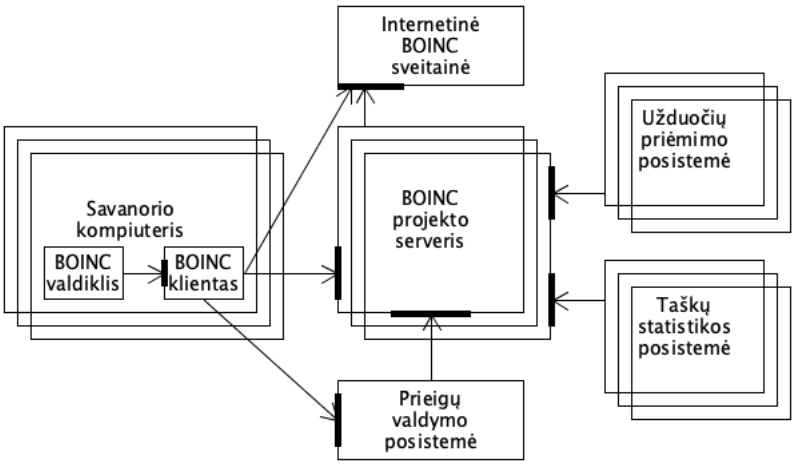
Yra įvairių viešai platinamų skaičiavimo sprendimų: „CharityEngine“[11], „GridMP“[12], „Xgrid“[13], „XtremWeb“[14]. Tačiau plačiausiai ir aktyviausiai naudojamas sprendimas vadinamas BOINC[15] („Berkeley Open Infrastructure for Network Computing“). BOINC yra didelio našumo plataus masto skaičiavimų platforma (skaičiavimuose gali dalyvauti tūkstančiai ar milijonai kompiuterių). Jis gali paleisti virtualizuotas, lygiagrečias, GPU skaičiavimais pagrįstas programas. Be to, jis gali vykdyti duomenų tyrybos užduotis naudodamas naudotojų įrenginius arba įmonės serverius [57]. BOINC skaičiavimus atlieka tik tada, kai centrinis procesorius (CPU) nėra aktyviai naudojamas. Šis sprendimas gali leisti organizacijoms naudoti turimus įmonės darbuotojų kompiuterinius išteklius netrukdydamas tuo metu vykstantiems darbo procesams. Kadangi įmonės darbuotojų kompiuterių procesoriai 99 %

[11] http://charityengine.com
[12] https://en.wikipedia.org/wiki/Grid_MP
[13] https://www.apple.com/server/docs/Xgrid_TB_v10.4.pdf
[14] http://xtremweb.gforge.inria.fr
[15] https://boinc.berkeley.edu

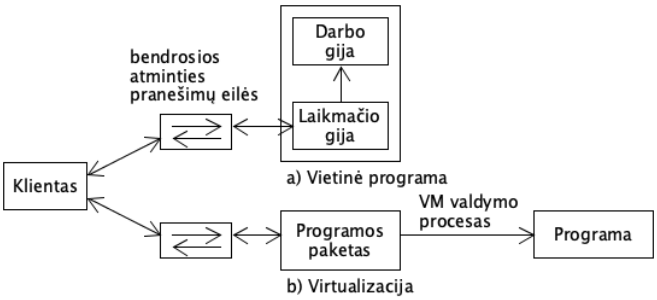
laiko yra mažai naudojami [57], šis sprendimas gali išspręsti skaičiavimo išteklių poreikio problemą.

2.1.1 Viešųjų paskirstytųjų skaičiavimų platforma BOINC

BOINC („Berkeley Open Infrastructure for Network Computing“) yra populiariausia ir beveik standartu laikoma savanoriškąja kompiuterija paremtų projektų programinė įranga [8, 44]. BOINC yra skirta savanoriškajai kompiuterijai ir skaičiavimams klasteriuose. Šis sprendimo būdas puikiai tinka tais atvejais, kai reikalinga ne tik pigi prieiga prie didelių skaičiavimo išteklių, bet ir kai vystomi projektai, turintys didelį visuomenės susidomėjimą atliekamais tyrimais [73]. Tai yra atvirojo kodo tarpinės programinės įrangos (angl. *middleware*) sistema, naudojanti paskirstytųjų skaičiavimų infrastruktūrą. Tokia infrastruktūra nepriklauso nuo mokslinių skaičiavimų ar eksperimentų pobūdžio. BOINC projektai paprastai yra skirti išspręsti sudėtingas mokslo problemas. Tokie projektai išnaudoja savanorių teikiamus išteklius, todėl projektų savininkai turi įgyti visuomenės pasitikėjimą ir susidomėjimą. Paprastai tai daroma pristatant projektą ir parodant jo pateikimą [68].



3 pav. BOINC sistemos komponentai ir RPC sąsajos



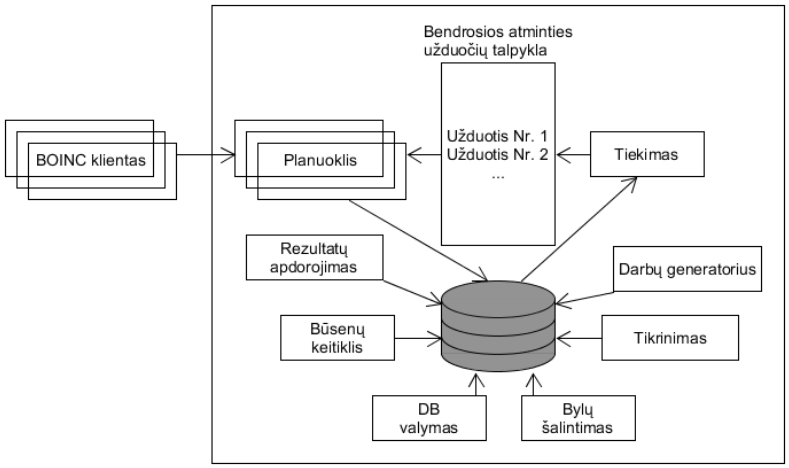
4 pav. BOINC vykdymosios aplinkos proceso struktūra

BOINC remiasi kliento-serverio architektūra ir gali būti naudojamas kaip pavyzdys, parodantis, kaip veikia viešųjų paskirstytųjų skaičiavimų modeliu grįstos platformos (žr. 3 ir 4 pav.):

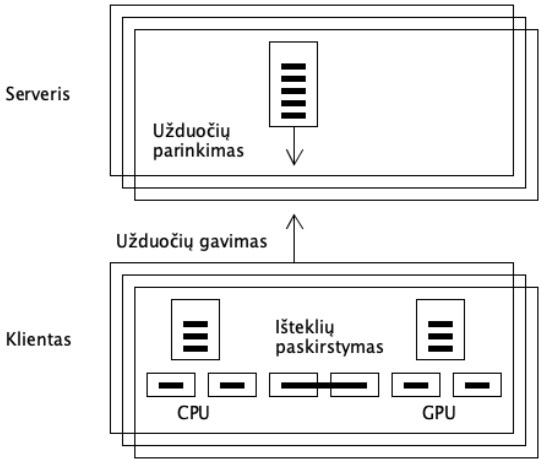
- duomenys saugomi bendroje duomenų bazėje;
- serveris dalija išteklių reikalaujančias užduotis į mažesnes;
- užduočių ir duomenų prašo ir juos gauna kliento mazgai;
- skaičiavimai atliekami be atskirų mazgų sąveikos;
- skaičiavimo rezultatai įkeliami į serverį ir sujungiami į galutinį sprendimą.

Norint geriau suprasti BOINC, galima peržvelgti teikiamų specialiųjų paslaugų, kurios paprastai veikia BOINC serveriuose, sąrašą (žr. 5 ir 6 pav.):

- būsenų keitiklis (angl. *transitioner*) – tvarko būsenos perėjimus, vienetus ir rezultatus;
- tiekimas (angl. *feeder*) – didina našumo planavimo programą;
- tikrinimas (angl. *validator*) – tikrina iš darbo padalinio gautų rezultatų pagrįstumą;
- rezultatų apdorojimas (angl. *assimilator*) – apdoroja rezultatus pagal konkrečiai programai skirtas taisykles;
- bylų šalinimas (angl. *file remover*) – naikina įvesties ir išvesties bylas, kai užbaigiami darbai;
- darbų generatorius (angl. *work generator*) – kuria darbo vienetus ir atitinkamus įvesties failus;
- duomenų bazės valymas (angl. *database cleaner*) – perkelia rezultatų ir darbo vienetų įrašus iš duomenų bazės į XML formato archyvo failus;
- tvarkaraščių planuoklis (angl. *scheduler*) – priskiria užduotis kliento mazgams pagal jų savybes.



5 pav. BOINC serverio komponentai



6 pav. BOINC darbų planavimas apima tris sąveikaujančias strategijas: išteklių paskirstymo, užduočių parinkimo ir užduočių gavimo

Centrinis serveris gali būti paskirstytas tarp kelių serverių sistemos apkrovai suvaldyti. BOINC API programuotojams ne tik leidžia paleisti savo BOINC programas BOINC platformoje ir sąveikauti su BOINC klientu [38], bet ir kurti ataskaitas, apdoroti vizualizacijas ir dėlioti kontrolinius taškus siekiant parodyti projekto būseną ir padėti išvengti darbo praradimo, kai skaičiavimo ištekliai tampa neprieinami. Taip pat yra sprendimų, leidžiančių senoms nebepalaikomoms (angl. *legacy*) programoms veikti BOINC klasteryje be jokių programinio kodo pakeitimų [66, 101].

Turimus išteklius galima skirti atsisiunčiant nedidelę (angl. *lightweight*) klientinę programą ir prijungiant ją per turimą projekto adresą (URL). Tokiu būdu BOINC klientas užmezga ryšį su BOINC serveriu, kontroliuoja programų ir darbo užduočių atsisiuntimą, kliento darbo planavimą ir rezultatų įkėlimą [12]. Savo ruožtu serveris filtruoja gautus klaidingus rezultatus iš nepatikimų išteklių, naudodamas pasirinktus patvirtinimo scenarijus (angl. *validation scripts*) arba dublikatų (angl. *redundancy*) paiešką [68].

Nepaisant savo populiarumo, BOINC vis dar turi daug trūkumų, kurie bus aptariami 2.1.4 skyriuje. BOINC taip pat turi du apribojimus [11]:

- BOINC platformoje veikiančios programos yra apribotos aplinkos (kompiuterių architektūros ir operacinės sistemos), kurioje yra vykdomos;
- BOINC klientas nesuteikia pakankamos apsaugos naudotojams.

BOINC pagrindu sukurtas darbinių kompiuterių klasteris suteikia galimybę mažoms mokslinėms grupėms ir mažoms bei vidutinėms įmonėms dirbti su didelio našumo reikalaujančiais skaičiavimais, pavyzdžiui, atliekant didžiųjų duomenų tyrybą. Tai galima padaryti naudojant savo arba savanorių teikiamus skaičiavimo išteklius. Gali pasirodyti, kad norint perduoti didelius duomenų kiekius analizei reikalingas išskirtinai didelis tinklo pralaidumas, tačiau remiantis [55], BOINC pagrindu sukurti darbinių kompiuterių klasteriai gali būti naudojami šiuolaikiniuose tinkluose. Tokie klasteriai suteikia galimybę apdoroti dešimtis terabaitų duomenų. Norint sujungti skaičiavimo išteklius, reikalingas greitas vietinis tinklas.

2.1.2 Debesų kompiuterija ir didelių duomenų tyryba naudojant BOINC

BOINC panaudojimo didelių duomenų tyrybai metodas gerai ištirtas [70]. Sukurta specialioji (angl. *ad hoc*) debesų kompiuterijos platforma, teikianti debesų kompiuterijos paslaugas galutiniams naudotojams priklausančioje infrastruktūroje, kurioje skaičiavimuose dalyvaujančių narių kompiuterių ištekliai yra ne visuomet prieinami ir yra naudojami kitam tikslui. Specialiosios debesų kompiuterijos koncepcija naudinga tiems, kurie nori pagerinti savo infrastruktūros efektyvumą ir panaudojimą, taip pat sumažinti išlaidas, taip padidinant IT investicijų grąžą. Be to, tie, kurie negali ar nenori pereiti prie komercinių ar privačių debesų kompiuterijos modelių, gali išbandyti ir ištirti specialiosios debesų kompiuterijos galimybes prieš priimdami sprendimą diegti bet kurį kitą komercinį ar privatų debesų kompiuterijos modelį. Specialioji debesų kompiuterija renka išteklius iš esamos, ne visuomet prieinamos, neišskirtinės (t. y. teikiančios prioritetą kitiems tikslams) ir nepatikimos infrastruktūros. Tokios infrastruktūros

pavyzdžiai svyruoja nuo asmeninių infrastruktūros naudotojų, turinčių keletą nepakankamai naudojamų kompiuterių, pradedančių įmonių iki plataus masto organizacinės infrastruktūros. Tyrimų iš [70] rezultatai rodo, kad specialiosios debesų kompiuterijos koncepcija yra įgyvendinama ir, remiantis pirminiu jų vertinimu, gali būti patikima ir pasiūlyti panašų našumą kaip ir „Amazon EC2“[16].

Be to, [90] integravo įrankius, tokius kaip „RapidMiner“[17], skirtus duomenų tyrybos užduotims vykdyti naudojant BOINC. Kiekvienas mazgas naudoja BOINC klientą, kad iš centrinio serverio gautų skaičiavimo užduotis ir duomenų rinkinius. Tada BOINC klientas paleidžia „RapidMiner“ sistemos karkasą duomenų tyrybos užduotims vykdyti. Duomenų tyrybos procese gauti rezultatai siunčiami atgal į centrinį projekto serverį, kuris kaupia informaciją ir teikia ją duomenų analitikams tolimesnei analizei atlikti.

Tai aiškiai parodo, kad viešųjų paskirstytųjų skaičiavimų metodas gali konkuruoti su esamais debesų kompiuterijos sprendimais. Kituose dviejuose poskyriuose apžvelgiami šio metodo pranašumai ir keliamos problemas.

Didelių duomenų tyryba yra naujų žinių iš didelių duomenų apimties gavimo būdas statistiniais metodais ar dirbtinio intelekto priemonėmis. Tai svarbus procesas daugelyje pramonės sričių: automobilių, sveikatos apsaugos, bankininkystės, draudimo, vartojimo prekių, naftos ir dujų, energijos ir komunalinių paslaugų, mažmeninės prekybos, vyriausybės, telekomunikacijų, kelionių ir transporto [55]. Tikimasi, kad visi pasaulio duomenys pasieks 39 trilijonų gigabaitų iki 2020 m. [25]. Nors mokslinių tyrimų, patvirtinančių šią prognozę, nėra, tačiau atlikta privačių tyrimų[18], teigiančių, kad ši prognozė pasitvirtino. Remiantis apklausos rezultatais, 2013 m. apie 28 % apklaustųjų, dirbančių duomenų tyrybos srityje, nagrinėjo rinkinius nuo 1 iki 100 petabaitų. Apie 2 % dirbo su duomenų rinkiniais, didesniais kaip 100 PB [84].

Didžiųjų duomenų tyrybos užduotims išspręsti reikalingos didelio našumo skaičiavimo sistemos. Norint gauti reikiamus išteklius, galima rasti keletą egzistuojančių sprendimų [55]:

- kompiuterių klasteriai (angl. *computing clusters*; įgyvendinimas ir palaikymas reikalauja didelių išlaidų);
- aptarnavimo klasteriai (angl. *service grids*; pigesni, tačiau palaikymas reikalauja pastangų);

[16] https://aws.amazon.com/ec2/
[17] https://rapidminer.com
[18] https://techjury.net/blog/big-data-statistics/

- stacionarių kompiuterių klasteriai (angl. *desktop grids*; pigūs ir lengvai palaikomi; surenka turimus išteklius iš asmeninių kompiuterių; nėra tokie patikimi).

BOINC gali būti laikomas savanoriškąją kompiuteriją naudojančių projektų vykdymo standartu. Tai pati populiariausia nemokama tarpinės programinės įrangos programinė įranga, naudojama heterogeniniams (angl. *heterogeneous*) stacionarių kompiuterių klasteriams [55]. Nors BOINC leidžia sukurti pigius įmonių lygmens skaičiavimo klasterius, kyla sunkumų kuriant BOINC pagrindu pagrįstas didžiųjų duomenų tyrybos programas [14, 55]:

- gali būti sunku suskaidyti užduotis į mažesnes, nepriklausomas užduotis;
- didžiuliai duomenų rinkiniai, kurie turi būti perduoti kliento mazgams, gali perkrauti tinklą;
- sunku pritaikyti programas dirbant su įvairiomis operacinėmis sistemomis ir architektūromis;
- naudotojų paskyrų priežiūra, dublikatų (angl. *redundancy*) ir programų klaidų šalinimas užima daug laiko.

Toliau nagrinėsime BOINC sistemą ir jos panaudojimo didžiųjų duomenų tyrybos uždavinių sprendimui būdus. Ypatingas dėmesys bus skiriamas našumui ir teikiamų paslaugų kokybei, nes debesų kompiuterijos paslaugos ir kiti alternatyvūs sprendimai gerai sprendžia šias problemas.

2.1.3 Duomenų tyrybos įrankiai, skirti BOINC

Vienas iš geriausiai žinomų būdų, kaip neapdorotus duomenis paversti vertinga informacija, yra „MapReduce“. Šis metodas skirtas apdoroti didelius duomenų rinkinius ir atlikti skaičiavimus, kai informacija išdėstyta dalimis keliuose kompiuteriuose. Šis metodas yra tapęs ir modeliu, pagal kurį kuriamos su didžiaisiais duomenimis dirbančios programos.

„MapReduce“ susideda iš dviejų svarbiausių dalių. Viena dalis duomenis rūšiuoja, filtruoja ir suskirsto pagal kategorijas, kad juos būtų galima analizuoti. Kita dalis sujungia duomenis ir pateikia jų santrauką. Šį metodą naudojo „Google“, tad jis tapo toks patikimas, kad jo pavadinimas netgi virto bendriniu terminu, apibūdinančiu bendrą modelį, kurį dabar naudoja įvairios technologijos.

[18] atliktas pastebėjimas rodo, kad dauguma platformų, kurios naudoja viešai internete prieinamus išteklius, sukurtos ir optimizuotos vykdyti uždavinių rinkinius (angl. *bag-of-tasks*). Todėl tokie sprendimai kaip BOINC,

„GridBot“ [93], „Bayanihan“ [89] ir daugelis kitų [3, 13, 23, 64] nepalaiko „MapReduce“ užduočių vykdymo. Vis dėlto atlikta tam tikrų tyrimų, kaip išspręsti šią problemą. Vienas iš tokių – BOINC-MR [28], sistema, galinti paleisti „MapReduce“ programas naudodama BOINC platformą. Tikslas buvo palaikyti „MapReduce“ („Google“ išpopuliarintą programinę įrangą [31]), esant nesaugiai ir nepatikimai aplinkai. Pateiktas sprendimas veikia taip:

1. klientas (reduktorius; angl. *reducer*) prašo naujos užduoties iš projekto serverio;
2. planavimo priemonė prie kiekvienos mažinimo užduoties prideda IP ir prievadą kompiuterių, kuriuose saugoma to paties darbo išvestis;
3. reduktorius turi galimybę atsisiųsti reikiamus įvesties failus tiesiai iš sąsajų sudarytojų (angl. *mappers*). Duomenų kopija yra saugoma, kad būtų galima išvengti klaidų;
4. kiekvienas reduktorius apdoroja atsisiųstus duomenis vykdydamas savo užduotį;
5. rezultatai siunčiami atgal į serverį.

Verta paminėti, kad BOINC-MR klientas gali veikti kaip reduktorius arba sąsajų sudarytojas (angl. *mapper*). Tai priklauso nuo gautos užduoties, o programa vykdoma paleidus reikiamą vykdomąjį failą [28].

„distribDataMining.org“ yra dar vienas projektas, integruojantis „RapidMiner“ į BOINC [90]. Projekto tikslas – tas pats: atlikti duomenų tyrybos užduotis. Tačiau sprendimas nėra toks sudėtingas. Duomenys su priskirtomis užduotimis paimami iš serverio naudojant BOINC klientą. Tada BOINC klientas vykdo duomenų tyrybos užduotis paleisdamas „RapidMiner“ sistemos karkasą. Klientas atlieka intensyvius lygiagrečius duomenų skaičiavimus ir siunčia rezultatus atgal į serverį. Rezultatai renkami ir siunčiami duomenų analitikams tolimesnei analizei.

„MapReduce“ darbo eigą būtų galima patobulinti priverčiant projekto duomenų paskirstymo posistemes naudoti tarpusavio duomenų dalijimosi protokolą, vadinamą „BitTorrent“ [18]. Jis leistų mazgams atsisiųsti duomenis iš kelių šaltinių vienu metu. Tai pagreitintų atsisiuntimo procesą, leistų padidinti galimų atlikti darbų mastą ir perkeltų dalį apkrovos iš centrinio serverio į skaičiavimus vykdančius mazgus. Nebepasiekiami mazgai duomenų perdavimo metu nepakenktų užduočiai, taigi padidėtų galimų gedimų toleravimas.

Tiesioginis metodas, kuris naudodamas BOINC pagrindu veikiantį įmonėms skirtą darbinių kompiuterių klasterį (angl. *enterprise desktop grid*) gauna sąsajas iš didelių duomenų rinkinių, aprašytas [55]. Jų sprendimas galėtų būti plečiamas ta pačia linkme, įgyvendinant kitus didžiųjų duomenų tyrybos metodus. Tai leistų mažoms, vidutinio dydžio įmonėms,

mokslininkams ir organizacijoms platformą naudoti savo didžiųjų duomenų tyrybos užduotims spręsti. Tačiau tokiu atveju kiekvienas algoritmas turės būti pritaikytas ir įdiegtas dirbti su BOINC.

Kitas įdomus metodas vadinamas vienkartine (lot. *Ad hoc*) debesų kompiuterija [70]. Ši platforma taip pat leidžia atlikti didžiųjų duomenų tyrybos užduotis naudojant BOINC, tačiau ji naudoja esamą naudotojų infrastruktūrą kaip debesų kompiuterijos paslaugą. Infrastruktūros mastas gali būti įvairus: pradedančioji įmonė, turinti kelis asmeninius kompiuterius, ar didelė organizacija. Skirtumas nuo standartinių debesų kompiuterijos paslaugų yra tas, kad vienkartinės debesų kompiuterijos pagrindu veikianti infrastruktūra išteklius gauna iš asmeninių kompiuterių. Šiuos kompiuterius jų savininkai taip pat gali naudoti kitoms užduotims spręsti ir jie retkarčiais gali būti neprieinami. Ši koncepcija pagerina infrastruktūros efektyvumą, turimų išteklių panaudojimą ir IT investicijų grąžą, nes sumažina teikiamų paslaugų kaštus. Tačiau šis sprendimas taip pat gali sukurti nepatikimą infrastruktūrą. Nepaisant to, šis sprendimo būdas yra labai naudingas naudotojams, norintiems pereiti prie komercinių ar privačiųjų debesų kompiuterijos modelių, tačiau neturinčių tam finansinių galimybių.

Rekomenduojama išbandyti ir ištirti vienkartinės debesų kompiuterijos galimybes, norint išsiaiškinti, ar toks sprendimas būtų tinkamas. Jei ne, tuomet gali būti priimtas sprendimas pereiti prie komercinio modelio. Remiantis pirminiu vertinimu, ši koncepcija yra įgyvendinama, ja paremti sprendimai – patikimi ir gali pasiūlyti su „Amazon EC2“ palyginamą našumą [70].

Apžvelgtos didžiųjų duomenų tyrybos platformos rodo, kad viešųjų paskirstytųjų skaičiavimų modeliu pagrįsti sprendimai gali konkuruoti su esamais debesų kompiuterijos sprendimais. Nors naudotojai renkasi technologijas, neturėdami su tuo susijusių profesionalių įgūdžių, naujausias debesų kompiuterijos modelis taip pat netinka mokslinėms problemoms spręsti. Sudėtingos problemos reikalauja didelės skaičiavimo galios ir atitinkamai – didelių finansinių investicijų. Abiem atvejais technologijos yra žinių ir sunkaus darbo derinys. Naudotojai, kuriems reikia išspręsti tam tikrą užduotį naudojant kokią nors naują technologiją, nenori investuoti daug laiko ir pastangų, kad suprastų, kaip ji veikia [11, 12].

2.1.4 BOINC diegimo problemos

Skirtingai nuo debesų kompiuterijos, viešųjų paskirstytųjų skaičiavimų modeliu grįstos paslaugos nėra tokios patikimos. Akivaizdu, kad taip yra dėl modelio pobūdžio, nes skaičiavimuose dalyvaujantys kompiuteriai bet kuriuo metu gali tapti neprieinami ir nėra galimybės žinoti, kiek skaičiavimo išteklių

bus pasiekiami. Bet kuriuo metu gali būti suaktyvinami bet kokio tipo ir dydžio duomenų apdorojimo uždaviniai. Tai gali būti ir svarbių duomenų apdorojimo užduotys, teikiamos skubos tvarka. Todėl nėra galimybės nustatyti, kiek laiko užtruks tam tikra užduotis. Taip pat nežinoma, ar vienu metu turimų išteklių pakanka, kad darbas būtų atliktas per reikiamą laiką. Toks darbo pobūdis verslo aplinkoje, kur nustatyti griežti projektų atlikimo tvarkaraščiai, nėra priimtinas. Atsižvelgiant į duomenų tipą ir skubą siųsti juos apdorojimui, reikia užtikrinti teikiamų paslaugų kokybę (angl. *quality of service*, QoS). Vienas iš būdų išspręsti šią problemą – sukurti matematinį modelį, kuris gebėtų įvertinti darbui reikalingą laiką ir kaštus. Tai gali būti įdomi būsimų tyrimų kryptis.

Kokybės gerinimo metodai ištirti [70]. Siūlomas metodas naudojasi virtualiųjų mašinų teikiamais pranašumais. Jų sprendimas vadinamas V-BOINC. Ši platforma sprendžia užduoties tęstinumo problemą. V-BOINC serveris siunčia virtualiosios mašinos atvaizdą ir scenarijų, kuris atlieka reikiamas konfigūracijas V-BOINC klientų kompiuteriuose (pvz., nustato procesoriaus, atminties ir disko vietos apribojimus). Tada virtualioji mašina sukonfigūruojama ir gali priimti BOINC užduotis bei grąžinti užduočių rezultatus. Tačiau tai kelia nerimą dėl saugumo. Anot [74], virtualiosioms mašinoms (VM) gresia pavojus net jei jos išjungtos. VM atvaizdus galima sugadinti į juos įskiepijant kenkėjiškus programinius kodus. Kita su VM atvaizdais susijusi problema yra ta, kad tokie atvaizdai gali išlaikyti pirminę informaciją apie buvusius jos savininkus, o ja gali pasinaudoti kiti naudotojai.

Norint, kad naudotojai ir įmonės diegtų viešųjų paskirstytųjų skaičiavimų modelio pagrindu veikiančias paslaugas ir jas naudotų, pirmiausia reikia išspręsti saugumo problemas. Aplinka, kurioje atliekami skaičiavimai, turi būti ne mažiau patikima nei debesų kompiuterijos pagrindu veikiančių paslaugų. Patikima aplinka – pagrindinė sąlyga norint įgyti naudotojų pasitikėjimą šia technologija. Labai svarbu apsaugoti duomenis nuo bet kokios neteisėtos naudotojų prieigos ar bet kokio kito išpuolio. Pasak [71], yra daug kriptografinių algoritmų, kuriuos galima diegti debesų kompiuterijoje, kad būtų užtikrintas saugumas. Toks pat požiūris gali būti taikomas ir viešiesiems paskirstytiesiems skaičiavimams. Pagrindinės debesų kompiuterijos savybės yra šios:

- paslaugos pagal pareikalavimą (angl. *on-demand self-service*);
- plati tinklo prieiga;
- išteklių telkimas (angl. *resource pooling*);
- greitas elastingumas (angl. *rapid elasticity*);
- paslaugų būsenos stebėjimas (angl. *measured service*) [71].

Visa tai jau yra prieinama BOINC, išskyrus išmatuotas paslaugas. Tai viena iš papildomų tyrimų reikalaujančių sričių.

Iš [71] ir kitų susijusių tyrimų matyti, kad debesų kompiuterija ir viešieji paskirstytieji skaičiavimai turi saugumo problemų. Pagrindinės debesų kompiuterijos saugumo problemos yra prieinamumas, duomenų ir sistemos vientisumas, autentifikavimas, duomenų atkūrimas, duomenų konfidencialumas, privatumas ir prieigos kontrolė. Tokias pačias problemas reikia spręsti viešųjų paskirstytųjų skaičiavimų modelyje. Tai verslui padarytų viešųjų paskirstytųjų skaičiavimų modeliu grįstus sprendimus prieinama alternatyva debesų kompiuterijos teikiamoms paslaugoms.

Savanoriškoji kompiuterija taip pat turi keletą iššūkių: nepastovumas, pasitikėjimo stoka, nenumatyti trikdžiai, heterogeniškumas, savanorių pritraukimas. Pereinant nuo brangių ir modernių debesų duomenų centrų prie savanorių išteklių, kyla tam tikrų iššūkių [12, 68, 73]:

- savanorių skatinimas tiekti išteklius ir kai kuriais atvejais suteikti daugiau prieigos prie savo kompiuterių;
- savanorių išteklių nepastovumo ir prieinamumo problemų sprendimas;
- paslaugų migracijos efektyvumo didinimas (reikia numatyti savanorių mazgų išteklių prieinamumą);
- naujų skaičiavimo išteklių ir vietos duomenims saugoti poreikio sprendimas;
- išteklių paskirstymas pagal poreikį;
- išlaidų racionalizavimas.

Paskelbta daugybė mokslinių tyrimų, kuriuose nagrinėjamos šios problemos ir nagrinėjamas savanorių ir debesų kompiuterijos derinimas, kaip galimi jau egzistuojančių projektų patobulinimai ar išplėtimai.

Remiantis [100], yra daugybė kitų veiksnių, darančių įtaką sprendimui įdiegti šią technologiją: aplinkosauginiai, organizaciniai, vadybiniai ir technologiniai. Šiame skyriuje apžvelgti tik technologiniai veiksniai. Kiti veiksniai gali sukelti papildomų diegimo problemų, kurios gali būti įdomios tyrimų kryptys ir atskleisti naujų sprendimams įtakos turinčių veiksnių.

Toliau apžvelgiami šiame skyriuje pateikti galimi problemų sprendimai.

2.1.5 Virtualizacija

Iš anksto įdiegti reikiamas bibliotekas ir patenkinti kitus reikalavimus įprastoms BOINC programoms – neįmanoma arba sunkiai išsprendžiama problema [68]. Virtualizacija naudojama norint paslėpti sistemos fizinius

išteklius nuo operacinės sistemos ir leidžia išspręsti daugelį su tuo susijusių problemų [69]. Virtualizacija apibrėžiama kaip technologija, sukurianti programinės įrangos abstrakcijos sluoksnį tarp aparatinės įrangos ir operacinės sistemos bei joje veikiančių programų [11, 88]. Dėl virtualizacijos technologijos programų perkėlimas į savanorių darbo kompiuterių klasterius tampa daug lengvesnis [68]. Kitu atveju, norint, kad programa veiktų skirtingose architektūrose, projekto kūrėjai privalo:

- kompiliuoti programą tikslinėms architektūroms;
- išsaugoti vykdymo eigą nutraukus programą;
- nenaudoti priklausomybių arba, jei įmanoma, įkompiliuoti priklausomybes į programą;
- įgyti savanorių pasitikėjimą, įtikinant, kad projektas yra patikimas ir neturi kenkėjiškų tikslų.

Tokių problemų sprendimas reikalauja daug projekto kūrėjų laiko. Virtualizacija pašalina nereikalingą perkėlimo į kitas architektūras žingsnio taikymą BOINC sistemoms. Geras pavyzdys yra parametrų valymo programos (angl. *parameter sweep applications*) [68]. Be virtualizacijos kiekvienai parametrų valymo programai prireiktų perkėlimo pastangų, taigi projekto vystytojams nebūtų įdomūs tokie infrastruktūros išplėtimo sprendimai, naudojantys asmeninių kompiuterių klasterius. Anksčiau BOINC platformos pagrindu veikiančias programas reikėdavo paruošti kiekvienai skirtingai kliento operacinei sistemai. Vykdant diegimus CERN [19, 46] (projekto „Test4Theory LHC@home“ iniciatorius 2010–2011 m. [19, 46, 50]), numatyta galimybė virtualiąją mašiną [94] paskirstyti savanorių kompiuteriams per BOINC [11]. „CernVM“ [91] pasitelkė „BOINC VBoxWrapper“ įrankį [11]. Vėliau BOINC kūrėjai įgalino virtualiosios mašinos funkcionalumą, įdiegdami sąsają („VBoxWrapper“) tarp BOINC kliento ir „VirtualBox“ [87]. Skaičiavimų vykdymui jie integravo „VirtualBox“ programinę įrangą, kurios naudojamų diskų atvaizduose saugojo skaičiavimus atliekančią programą ir jos duomenis. Duomenis ir rezultatus saugojo bendro naudojimo aplanke tarp pagrindinės kompiuterio operacinės sistemos ir virtualiosios mašinos.

Be to, [69] pristatė dar vieną projektą, pavadintą V-BOINC. V-BOINC taip pat naudoja virtualizaciją, kad paleistų programas savanorių kompiuteriuose, ir siunčia mažus virtualiosios mašinos vaizdus savanoriams. Tokiu būdu BOINC programos veikia virtualiosiose mašinose, o ne tiesiogiai pagrindinėje operacinėje sistemoje. V-BOINC savo ruožtu leido kūrėjams patenkinti programų reikalavimus ir naudojamų bibliotekų priklausomybes, ir padidino programų, kurias gali vykdyti savanorių infrastruktūra, įvairovę. Tokiu būdu

atsirado galimybė įgyvendinti daugiau mokslo ir verslo projektų, nes paraiškos tapo prieinamos platesnei visuomenei [69]. Be to, V-BOINC išsprendė tokias papildomas problemas kaip:

- pasitikėjimas programa;
- programos būsenos išsaugojimas nutraukus programos vykdymą.

Nors vykdymo laikas yra ilgesnis dėl virtualizacijos (dėl virtualiosios aplinkos paleidimo ir hipervizoriaus [69]), daugelis savanorių skaičiavimo projektų naudojo V-BOINC. Verta paminėti, kad ne visi procesoriaus branduoliai ir atmintis gali būti naudojami dėl „VirtualBox“ apribojimų. Be to, virtualiųjų diskų atvaizdų dydis gali svyruoti nuo kelių šimtų megabaitų iki dešimčių gigabaitų. Dėl to virtualiųjų diskų atvaizdų valdymas ir dislokavimas taip pat yra didelis iššūkis [68].

2.1.6 Išteklių kiekio prognozavimas

BOINC klientas įvertina pagrindinės kompiuterio aparatinės įrangos pajėgumą ir charakteristikas periodiškai matuodamas pagrindinio kompiuterio prieinamumo parametrus, tokius kaip veikimo laiką ir periodą, kai klientas yra prisijungęs. BOINC taip pat reguliariai vykdo „Whetstone“ [29] ir „Dhrystone“ [108] vertinimus (angl. *benchmarks*) [9]. Tačiau BOINC matuoja atminties kiekį diske tik tuose įrenginiuose, kur yra įdiegtas. Dėl to gali būti pervertinama turima laisva atmintis tinklu prieinamuose diskuose, (angl. *shared network-accessible volumes*), kai daugiau kompiuterių naudoja BOINC klientą. Nepaisant to, atliktas potencialių savanorių kompiuterija paremtos platformos išteklių pajėgumų tyrimas. Tyrimai buvo pagrįsti galios, atminties, vietos diske, tinklo pralaidumos, kompiuterių išteklių prieinamumo, naudotojų nurodytų išteklių naudojimo apribojimų ir dalyvių kaitos (angl. *host churn*) vertinimu [9]. Tyrimas parodė, kad šių dalykų pakanka bendram savanorių teikiamų išteklių pajėgumui apskaičiuoti. Gauta išraiška (1) rodo bendrą projektui skirtą slankiojo kablelio skaičiavimo galią X [9]:

$$X = X_{arrival} \times X_{life} \times X_{ncpus} \times X_{flops} \times X_{eff} \times X_{onfrac} \times X_{active} \times X_{redundancy} \times X_{share}. \quad (1)$$

Kaip apibrėžta [9], $X_{arrival}$ – vidutinė naujų kompiuterių dalis, X_{life} – vidutinis kompiuterių eksploatavimo laikas, X_{ncpus} – vidutinis CPU skaičius vienam kompiuteriui, X_{flops} – vidutinis slankiojo kablelio operacijų kiekis per sekundę vienam procesoriui, X_{eff} – vidutinis CPU efektyvumas, X_{onfrac} – vidutinė laiko trukmė, kai kompiuteris įjungtas, X_{active} – vidutinė laiko trukmė,

kai kompiuteris aktyvus, $X_{redundancy}$ – vidutinio pasitraukimo iš darbo dalis, o X_{share} – vidutinė išteklių dalis (palyginti su kitais procesoriams imliais projektais).

Tačiau šis sprendimas gali būti nepakankamas. Prognozavimo metodikos gali žymiai pagerinti savanorių išteklių paskirstymo valdymo efektyvumą. Tai galima padaryti įvertinant trumpalaikius išteklių elgsenos modelius. „Climateprediction.net“ projektas (CPDN) sukurtas 1999 m. (Allen, 1999; CPDN, 2015) kaip paskirstytųjų skaičiavimų iniciatyva siekiant pašalinti išteklių kiekio neaiškumus [73]. Skubiam modeliavimui realiuoju laiku ir netikėtiems įvykiams gali prireikti daugiau išteklių nei tuo metu gali suteikti savanorių kompiuteriai. Negalima tiksliai numatyti ar išmatuoti savanorių elgesio, tačiau prognozė padeda imtis tinkamų atsargumo priemonių [10]. Vienas iš būdų tai ištaisyti – perkelti savanorių skaičiavimų naštos dalį infrastruktūrai kaip paslaugai (angl. *infrastructure as a service*, IaaS), kuri būtų pagrįsta debesų kompiuterija (pvz., „Amazon Web Services“, AWS) [73].

1 lentelė: Vidutinis debesijos paslaugų pasiekiamumas [12]

Debesų kompiuterijos paslaugų teikėjas	Paslaugos pavadinimas	Paslaugos vidutinis pasiekiamumas per mėnesį
Google	Google apps	< 99,9 % -> = 99,0 %
Amazon	Amazon EC2	< 99,9 % -> = 99,0 %
Amazon	Amazon S3	< 99,9 % -> = 99,0 %
Microsoft	Cloud services	< 99,95 %
Cloud flare	CloudFlare	100 %

Organizacijos nerimauja dėl debesų kompiuterijos paslaugų prieinamumo [12]. Ne tik stacionarių kompiuterių klasteriai, tokie kaip BOINC ar „XtremWeb“ [21], turi centralizuotą architektūrą, sukeliančią potencialią kliūtį nuolatinėje savanorių skaičiavimo sistemų evoliucijoje, bet taip pat yra nerimą keliančių aktyvių naudotojų ir projektų sąstingio požymių. Tai sukelia problemų, susijusių su duomenų saugojimu ir platinimu [7, 27, 28]. Prognozuojant išteklių prieinamumą, paskirstytųjų skaičiavimų aplinkos naudojimo efektyvumas gali būti labai naudingas. Kaip parodyta 1 lentelėje, visi pagrindiniai debesų paslaugų teikėjai siūlo aukštą prieinamumą.

Pagrindinis asmeninių kompiuterių klasterio trūkumas yra išteklių nepastovumas. Tokias problemas lemia sistemos valdymo politika [10]. Automatinės elgesio numatymo sistemos gali padėti išspręsti šią problemą.

Tokias sistemas gali sudaryti kalendoriniai metodai [51] arba klasifikavimo algoritmai, žinomi iš duomenų tyrybos [110]. Pagrindinis šių metodų trūkumas yra tas, kad tiriamasis laikotarpis nustatomas subjektyviai. Pavyzdžiui, pažangiausia išteklių prognozavimo sistema kompiuterinėse sistemose – išteklių numatymo sistema (angl. *resource prediction system*) [32], tačiau pagrindinis trūkumas yra tas, kad joje daug dėmesio skiriama labai trumpalaikėms prognozėms. Prognozavimo tyrimas atliktas imant institucinių darbo kompiuterių imtį (angl. *institutional desktop pool*), naudojant tris numatymo tikslus: procesoriaus laisvosios eigos laiką, atminties apkrovą ir kompiuterio prieinamumą [10]. Tyrimo išvada parodė, kad net ir labai dinamiška aplinka, tokia kaip darbuotojų kompiuteriai, leidžia prasmingai numatyti įvairius rodiklius naudojant palaikymo vektorių mašinų klasifikatorių (angl. *support vector machines cassifier*; SMO).

2.1.7 Kaštų vertinimas

Savanoriškoji kompiuterija gali pastebimai sutaupyti energijos kaštų, nes sumažėja vidinių išteklių skaičiavimams atlikti poreikis. Tai leidžia įsigyti techninę įrangą iš daugelio naudotojų, taigi nukreipiamos išlaidos. Tai ypač svarbu, jei skaičiavimams naudojami GPU, kurie gali padidinti energijos poreikį iki 30 % kiekvienam kompiuteriui [44]. Dabartiniai aukščiausios klasės GPU sunaudoja labai daug energijos, todėl tai didelė problema, turintiems didelius klasterius. Energijos tiekimo išlaidos gali sudaryti didelę viso turto išlaidų dalį [36, 37].

Yra GPU pagrįstų infrastruktūrų sąnaudų įvertinimo modelių, naudojančių lygtis, kurios skaičiuoja kiekvieno kompiuterio sąnaudas [44]. Lygtys gali atsižvelgti į energijos suvartojimo sąnaudas, kompiuterių rinkos kainą ir kompiuterių sandėliavimo sąnaudas. Nepaisant tyrimų pastangų, vis dar nėra patikimo būdo nustatyti duomenų tvarkymo laiko sąnaudas, kuris padėtų numatyti turimų išteklių pajėgumus.

2.1.8 Išteklių prieinamumo ir energijos suvartojimo tyrimas

Paskirstytųjų skaičiavimų platforma, naudojanti BOINC karkasą, leidžia bet kuriai organizacijai sujungti turimus skaičiavimo išteklius į dinaminio dydžio skaičiavimų platformą, kuri atliktų duotas skaičiavimo užduotis ir netrikdytų darbuotojų atliekamo darbo. Toks sprendimas išsprendžia ne tik skaičiavimo išteklių poreikio problemą, bet ir duomenų konfidencialumo problemą, nes visi skaičiavimai atliekami organizacijos viduje. Jei reikia, organizacijos gali nustatyti norimus duomenų prieigos lygius. Be to, tokia

platforma gali padidinti organizacijos teikiamų paslaugų kokybę, nes leidžia atlikti sprendimus, pritaikytus verslo poreikiams. Galiausiai tai gali sumažinti paslaugų įkainius ir kaštus.

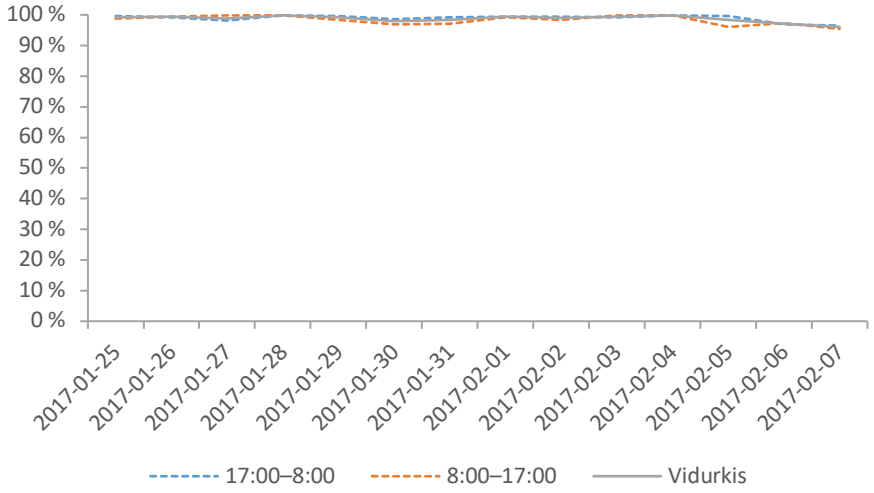
Siekiant įrodyti išteklių prieinamumo ir išlaidų sumažinimo teiginius atliktas 28 dienų trukmės eksperimentas naudojant du atsitiktinai pasirinktus kompiuterius iš skirtingų organizacijų (A ir B organizacija). Kiekviename kompiuteryje įdiegti BOINC klientai dvi savaites vykdė „SETI@home“[19] projekto skiriamas užduotis. Per tą laiką energijos suvartojimo ir procesoriaus laisvosios eigos matavimai atlikti naudojant „Performance Monitor“ (programa platinama kartu su „Microsoft Windows 10“ operacine sistema) ir elektroninį energijos matavimo įrenginį. Tada procesas pakartotas nevykdant jokių BOINC projektų. Eksperimentų metu darbuotojai abu kompiuterius naudojo atlikdami su darbu susijusias užduotis. Kaip parodyta 2 lentelėje, 7 ir 8 pav., kompiuteriai atlieka labai mažai skaičiavimų ir eikvoja turimus išteklius. Tyrimo rezultatai publikuoti [58].

2 lentelė: Išteklių ir elektros energijos suvartojimo statistika

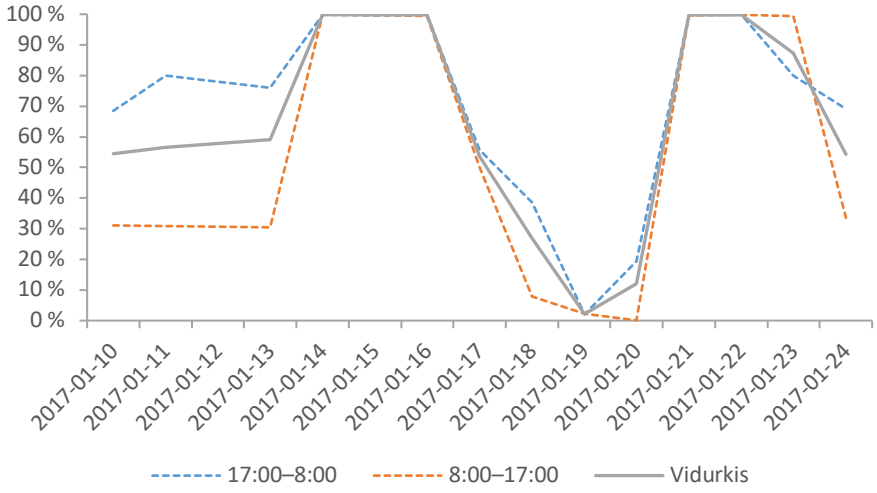
#	BOINC projektas	CPU vidutinis prastovos laikas	Elektros energijos suvartojimas
A	Nevykdoma	98,77 %	16,61 kWh
	SETI@home	65,23 %	22,03 kWh
B	Nevykdoma	83,49 %	1 kWh
	SETI@home	26,86 %	2,09 kWh

Kadangi duomenims rinkti buvo naudojamas tik nedidelis kompiuterių kiekis, reikėtų atlikti tolimesnius tyrimus, kuriuose dalyvautų daugiau kompiuterių. Nepaisant to, surinkti rezultatai jau rodo, kad infrastruktūrą galima naudoti papildomiems skaičiavimams už labai mažą kainą, nenutraukiant tuo metu vykdomų darbo procesų. Kadangi iš darbuotojų darbo kompiuterių gaunami ištekliai naudojami tik nedideliu mastu, įdiegus viešųjų paskirstytųjų skaičiavimų platformą tokia infrastruktūra gali būti naudojama papildomoms užduotims vykdyti (pavyzdžiui, didelių duomenų tyrybai).

[19] https://setiathome.berkeley.edu



7 pav. Procesoriaus tuščiosios eigos laikas nevykdant BOINC projekto A organizacijoje



8 pav. Procesoriaus tuščiosios eigos laikas vykdant SETI@home projektą A organizacijoje

„SETI@home“ projektas specialiai skirtas didelių duomenų rinkiniams tvarkyti. Pabrėžtina, kad vidiniai skaičiavimo ištekliai gali būti naudojami ir kitoms didžiųjų duomenų tyrybos užduotims. Kyla klausimas, kokio dydžio ir pajėgumų turėtų būti vidinė IT infrastruktūra tokioms užduotims atlikti.

Nors eksperimente dalyvavo nedaug kompiuterių, rezultatai rodo, kad toks sprendimas gali būti naudingas. Darbuotojai neišnaudojo visų kompiuterių išteklių, o tai leido atlikti papildomus skaičiavimus už nedidelę kainą. Be to,

atliekami skaičiavimai nesutrikdė nė vieno vykstančio darbo proceso. Organizacija gali atlikti papildomas didžiųjų duomenų tyrybos ir panašias užduotis naudodama viešųjų paskirstytųjų skaičiavimų platformą. Tačiau iš tolesnio eksperimento su kompiuterių imtimi turėtų būti galima padaryti daugiau įžvalgų.

2.2 Privačiųjų paskirstytųjų skaičiavimų platforma „Apache Mesos“

Viena iš klasterių išteklių valdymo platformų vadinama „Apache Mesos“ [49]. Ji palaiko populiarias sistemas, tokias kaip „Hadoop“[20] ir MPI[21], gali valdyti iki 50 000 (emuliuotų) mazgų ir turėti mažesnį kaip 4 % perviršį. Vis dėlto rekomenduojama teikti pirmenybę mažoms, o ne didelėms užduotims, kad būtų galima sumažinti laiko sąnaudas, atsirandančias dėl netikėtų trikdžių. „Apache Mesos“ palaiko įvairius užduočių planuoklius, tokius kaip „Apache Chronos“[22]. „Apache Chronos“ atsakingas už grafikų ir priklausomybių pagrindu vykdomų darbų planavimą. Tačiau vis daugiau neapdorotų užduočių gali sukelti užduočių planuoklio klaidą. Sukurtoje hibridinių paskirstytųjų skaičiavimų platformoje ši problema išspręsta apribojant neapdorotų užduočių skaičių iki turimų išteklių skaičiaus „Apache Mesos“ klasteryje. Galiausiai, būtina pažymėti, kad „Apache Chronos“ ir „Apache Mesos“ reikalauja saugios tinklo aplinkos.

2.3 Hibridinių paskirstytųjų skaičiavimų platformos

Hibridinių paskirstytųjų skaičiavimų platformos sujungia privačiųjų ir viešųjų paskirstytųjų skaičiavimų klasterius, kad būtų padidintas atliekamų skaičiavimų našumas ir pagerintas sistemos patikimumas [96]. Paskirstytųjų skaičiavimų užduotys paskirstomos privačiųjų ir viešųjų skaičiavimų ištekliams taikant įvairius užduočių paskirstymo algoritmus (žr. 3 lentelę). Šių algoritmų tikslas – skaičiavimo ištekliams paskirstyti užduotis, pagerinant vieną ar daugiau šių sistemos optimizavimo kriterijų [92]:

- sutrumpinti užduoties vykdymo trukmę;
- sumažinti sistemos atsako laiką;
- sumažinti užduočių aibės vykdymo trukmę;
- padidinti sistemos našumą;

[20] https://github.com/mesos/hadoop
[21] https://github.com/mesosphere/mesos-hydra
[22] https://mesos.github.io/chronos/

- padidinti skaičiavimo išteklių išnaudojimo efektyvumą;
- pagerinti skaičiavimo išteklių apkrovos paskirstymą;
- pagerinti klaidų toleravimą;
- sumažinti energijos sunaudojimą;
- pagerinti apkrovos šuolių toleravimą;
- sumažinti skaičiavimo kaštus;
- pagerinti teikiamų paslaugų kokybę.

3 lentelė: Egzistuojančios hibridinių paskirstytųjų skaičiavimų platformos ir užduočių paskirstymo algoritmai

Tyrimas	Metodas	Tikslas
Topcuoglu, H. ir kiti (2002) [102]	Aukščiausio prioriteto užduotis priskiria greičiausiai rezultatus galintiems grąžinti skaičiavimo ištekliams.	Minimizuoti užduočių aibės įvykdymo trukmę.
Bittencourt, L. F. ir kiti (2011) [17]	Patobulintas HEFT [102] algoritmas.	Minimizuoti užduočių aibės įvykdymo trukmę, užtikrinti užduočių atlikimą per nustatytą terminą.
Bittencourt, L. F. ir kiti (2012) [16]	Optimaliam sprendimui surasti naudojamas tiesinis programavimas (angl. „*linear programming*“).	Minimizuoti užduočių aibės įvykdymo trukmę, užtikrinti užduočių atlikimą per nustatytą terminą.
Vecchiola, C., Calheiros, R. N. ir kiti (2012) [20, 104]	Sukurtas naujas užduočių paskirstymo ir priežiūros algoritmas. Lėtai vykdomoms užduotims priskiriama daugiau skaičiavimo išteklių.	Užtikrinti užduočių įvykdymą per nustatytą terminą.
Van den Bossche, R. ir kiti (2013) [103]	Sukurtos taisyklės, nustatančios sąlygas naudoti FIFO, EDF ar savo sukurtus užduočių paskirstymo algoritmus, paremtus skaičiavimo kaštų vertinimu.	Minimizuoti užduočių aibės įvykdymo trukmę, užtikrinti užduočių atlikimą per nustatytą terminą.

Duan, R. ir kiti (2014) [35]	Optimaliam sprendimui surasti naudojama žaidimų teorija.	Minimizuoti užduočių aibės įvykdymo trukmę, minimizuoti skaičiavimo kaštus.
Wang, B. ir kiti (2016) [105]	Sukurtas naujas užduočių paskirstymo algoritmas. Užduotys skaičiavimo ištekliams paskirstomos remiantis skaičiavimų kaštų ir skaičiavimų efektyvumų santykių vertinimais.	Maksimizuoti išteklių panaudojimą.
Zhang, Y. ir kiti (2017) [112]	Patobulintas PSO algoritmas.	Užtikrinti užduočių įvykdymą per nustatytą terminą, minimizuoti skaičiavimo kaštus.
Abdi, S. ir kiti (2017) [2]	Optimaliam sprendimui surasti naudojamas tiesinis programavimas.	Minimizuoti užduočių aibės įvykdymo trukmę, užtikrinti užduočių atlikimą per nustatytą terminą.
Zhang, Y. ir kiti (2019) [114]	Taikomas matematinio optimizavimo algoritmas „Firefly“.	Minimizuoti užduočių aibės įvykdymo trukmę, minimizuoti skaičiavimo kaštus.
Zhang, Y. ir kiti (2019) [113]	Sukurtas užduočių perskirstymo algoritmas.	Minimizuoti užduočių aibės įvykdymo trukmę, minimizuoti skaičiavimo kaštus.
Stavrinides, G. L. ir kiti (2021) [96]	Patobulinti Min-Min, Min-Max algoritmai [15, 34].	Užtikrinti užduočių įvykdymą per nustatytą terminą, minimizuoti skaičiavimo kaštus.

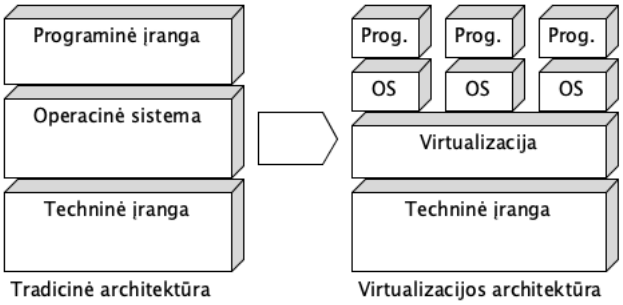
Ši apžvalga parodė, kad visuose darbuose pasitelkti užduočių paskirstymo algoritmai naudoja turimus išankstinius duomenis apie vykdyti pateiktų užduočių kiekį, kiekvienos užduoties vykdymo trukmę arba turimų skaičiavimo išteklių kiekį. Pagal šiuos duomenis sukuriamas užduočių

vykdymo tvarkaraštis. Tačiau heterogeniškuose paskirstytųjų skaičiavimų tinkluose šie parametrai nuolat kinta arba duomenų nėra. Todėl autorių darbuose siūlomi sprendimai negali būti taikomi.

Disertacijoje pristatoma nauja hibridinių paskirstytųjų skaičiavimų platforma minimizuoja užduočių aibės įvykdymo trukmę heterogeniškuose hibridinių paskirstytųjų skaičiavimų tinkluose. Siekiant sukurti tokią platformą reikalingas naujas užduočių paskirstymo metodas. Be to, svarbu atsižvelgti į užduočių įstrigimo paskirstytųjų skaičiavimų tinkluose problemą, kuri yra viena iš pagrindinių priežasčių, neleidžiančių užduočių vykdymui išnaudoti įmonių vidinių skaičiavimo išteklių [43]. Todėl 3.3 skyriuje apžvelgiami egzistuojantys hierarchiniai ir nehierarchiniai užduočių paskirstymo algoritmai, kurie galėtų būti pritaikyti hibridinių paskirstytųjų skaičiavimų platformose. Disertacijoje pristatomas naujas sukurtas užduočių paskirstymo algoritmas, nes tinkamiausias šiuo metu egzistuojantis užduočių paskirstymo algoritmas užduotis gali skirstyti tik tarp dviejų klasterių. Todėl 3.3.4 skyriuje pristatoma šio algoritmo modifikacija.

2.4 Programų virtualizacija

Programinės įrangos virtualizacija – technologija, paslepianti fizinius sistemos išteklius nuo operacinės sistemos ir padedanti išspręsti įvairias problemas [69]. Heterogeniškoje aplinkoje programinės įrangos virtualizacija leidžia vykdyti tas pačias užduotis keliose kompiuterių architektūrose ir skirtingose operacinėse sistemose. Skirtumas tarp tradicinės architektūros ir virtualizacijos pateiktas 9 pav.



9 pav. Programinės įrangos virtualizacijos architektūra

Yra daugybė įvairių programinės įrangos virtualizacijos technologijų: „Docker“ [23], „Kubernetes“ [24], „Oracle VM VirtualBox“ [25], „QEMU“ [26], „VMware“ [27] ir daugelis kitų. Programinės įrangos suderinamumui su „Apache Mesos“, „Apache Chronos“ ir BOINC naudojami „Docker“ ir „Oracle VM Virtual Box“.

„Docker“ – platforma, teikianti paslaugų rinkinį (angl. *platform as a service*). Ši platforma naudoja operacinės sistemos lygio virtualizaciją ir suteikia galimybę susieti programinę įrangą paketuose, vadinamuose konteineriais (mažesni už virtualiosioms mašinoms naudojamus diskų atvaizdus). „Docker“ leidžia paruošti programinę įrangą taip, kad ją būtų galima vykdyti įvairiose kompiuterių architektūrose ir operacinėse sistemose. Tačiau tai reikalauja, kad užduotis vykdančios programos taip pat palaikytų jas vykdysiančias operacines sistemas ir procesorių architektūras.

„Oracle VM VirtualBox“ programa skirta kurti, valdyti ir paleisti virtualiąsias mašinas. Ji suteikia aparatūros lygio virtualizaciją ir turi daugiau saugos valdiklių už „Docker“. Taip pat aparatūros lygio virtualizacija leidžia paleisti programinę įrangą įvairiose kompiuterių architektūrose ir operacinėse sistemose be pakeitimų poreikio užduotis vykdančiose programose. Tačiau virtualiosios mašinos naudoja daugiau kompiuterio išteklių ir paleidimas trunka ilgiau nei naudojant konteinerius.

Nors programinės įrangos virtualizacija taip pat gali išspręsti kai kurias debesų kompiuterijos saugumo problemas, ji neapsaugo nuo visų saugumo grėsmių.

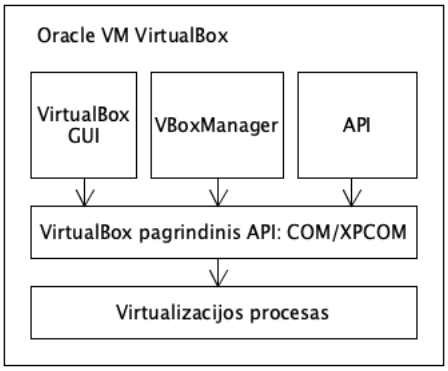
2.4.1 „VirtualBox”

„VirtualBox“ yra nemokamas atviro kodo hipervizorius, skirtas „x86“ ir „AMD64“ procesorių architektūrų virtualizacijai. Naujausia „VirtualBox“ 6.1 versija palaiko daugybę operacinių sistemų, įskaitant „Windows“ (NT 4.0, 2000, XP, „Server 2003“, „Vista“, „Windows 7“, „Windows 8“, „Windows 10“), „Linux“ (2.4, 2.6, 3.x, 4.x, 5.x) ir „Mac OS X“.

„Oracle VM VirtualBox“ leidžia virtualiojoje mašinoje esančią programinę įrangą paleisti naudojant fizinio kompiuterio procesorių. Dalis aparatinės įrangos, su kuria virtualiojoje mašinoje veikianti programinė įranga

[23] https://www.docker.com
[24] https://kubernetes.io
[25] https://www.virtualbox.org
[26] https://www.qemu.org
[27] https://www.vmware.com

gali sąveikauti, yra imituojama. Tai leidžia „Oracle VM VirtualBox“ imtis veiksmų, jei kenkėjiška programinė įranga siekia pakenkti kompiuteriui.



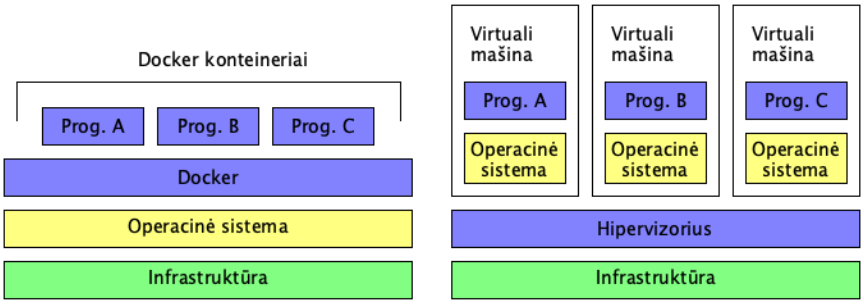
10 pav. Sąsajos sąveikai su „Oracle VM VirtualBox“

„Oracle VirtualBox“ naudotojo sąsają sudaro šie komponentai (žr. 10 pav.):

- „VirtualBox“ GUI – grafinė naudotojo sąsaja;
- „VBoxManager“ – naudotojo sąsaja, prieinama naudojant komandinę eilutę;
- API (angl. *application programming interface*) – programos programavimo sąsaja;
- „VirtualBox“ pagrindinis API – programavimo sąsaja, jungianti visas naudotojo sąsajas ir programinės įrangos komponentus.
    - COM (angl. *component object model*) – binarinės sąsajos programinės įrangos komponentų standartas, naudojamas komunikacijai tarp procesų „Windows“ operacinėje sistemoje;
    - XPCOM (angl. *cross platform component object model*) – binarinės sąsajos programinės įrangos komponentų standartas, naudojamas komunikacijai tarp procesų kitose nei „Windows“ operacinėse sistemose;
- Virtualizacijos procesas – „VirtualBox“ programos pagrindinis procesas.

2.4.2 „Docker“

„Docker“ yra konteinerių grįsta virtualizacijos technologija. Konteineriai – nuo likusios sistemos izoliuoti procesų rinkiniai. Procesai paleidžiami naudojant atskirus diskų atvaizdus, o kiekviename atvaizde pateikiami visi procesui reikalingi failai. „Docker“ programoje veikiantys konteineriai procesams vykdyti naudoja fiziniame kompiuteryje veikiančios operacinės sistemos branduolį.



11 pav. „Docker“ konteineriai

„Docker“ konteineriais paremtos programų virtualizacijos architektūros lyginimas su virtualiosiomis mašinomis grįsta architektūra pateiktas 11 pav. „Docker“ palaiko įvairias procesorių architektūras ir įvairias operacines sistemas, tačiau konteineriai ir juose vykdomos programos yra tiesiogiai priklausomos nuo fizinio kompiuterio procesoriaus architektūros bei jame veikiančios operacinės sistemos. Dėl to konteineriuose veikiančios programos privalo būti atitinkamai paruoštos kiekvienos galimos sistemos ir procesoriaus architektūros atvejui atskirai.

2.5 Saugumo problemos

Sistemų ir informacijos saugumas visada kėlė susirūpinimą. Šias problemas būtina spręsti tam, kad naudotojai ir įmonės bent jau apsvarstytų galimybę įdiegti ir naudoti viešuosius paskirstytuosius skaičiavimus. Į panašias saugumo problemas atkreiptas dėmesys ir jos sėkmingai išspręstos debesų kompiuterijos modelyje, kuris šiuo metu yra plačiai naudojamas. Tai nenuostabu, nes vienas iš pagrindinių bet kurios informacinės sistemos reikalavimų yra saugumas. Naudotojai turi būti tikri savo apsauga ir patikėti savo duomenis ir išteklius naudojamai aplinkai. Duomenys turi būti apsaugoti

nuo bet kokios neteisėtos prieigos, įskaitant išpuolius ir bandymus įsilaužti į sistemą.

Kriptografija – vienas iš būdų, galinčių padėti apsaugoti duomenis. Daugybė kriptografinių algoritmų gali būti naudojami debesų kompiuterijoje, užtikrinant didesnį saugumą [71]. Toks sprendimas taip pat gali būti įdiegtas viešųjų paskirstytųjų skaičiavimų programose. Remiantis [71], debesų kompiuterijos sprendimai gali pasiūlyti savitarnos paslaugas ir išteklių telkimą (angl. *resource pooling*). Infrastruktūra turi palaikyti greitą elastingumą ir turėti plačią prieigą prie tinklo. Galiausiai paslauga ir jos išlaidos turi būti išmatuojamos. BOINC jau atitinka daugumą šių reikalavimų, įskaitant turimų išteklių ir sistemos apkrovos įvertinimo galimybes. Tačiau nepavyksta išmatuoti išlaidų ir rezultatų gavimo laiko. Šie apribojimai reikalauja tyrimų ir juos reikia išspręsti.

Tiek viešųjų paskirstytųjų skaičiavimų, tiek debesų kompiuterijos modeliai kelia panašų susirūpinimą dėl saugumo, kuris apima [71]:

- sistemos prieinamumą;
- duomenų ir sistemos vientisumą;
- naudotojo autentifikavimą;
- duomenų atsarginių kopijų kūrimą ir duomenų atkūrimą;
- duomenų konfidencialumo užtikrinimą;
- privatumo ir prieigos kontrolę.

Paskirstytųjų skaičiavimų ir debesų kompiuterijos modeliai turi daug bendrų problemų. Tačiau daugelis iš jų jau išspręstos debesų kompiuterijos modelyje. Viešieji paskirstytieji skaičiavimai yra pigus sprendimas sprendžiant išteklių reikalaujančias užduotis. Išsprendus šias problemas, jie taptų gera alternatyva debesų kompiuterijos paslaugoms ir galėtų atverti daug naujų galimybių.

Pagrindinis savanoriškosios kompiuterijos iššūkis – saugumas, nes visi skaičiavimo darbai atliekami naudojant savanorių išteklius [12]. BOINC naudojasi dviem mažiau privilegijuotomis prieigomis. Pirmoji, turinti daugiau privilegijų, skirta BOINC klientui. BOINC klientas nuolat stebi veikiančias programas, kurios vykdomos naudojant griežtesnę prieigą. Tačiau, anot [68], kenksmingos programos gali išvengti šios priežiūros. Debesų kompiuterijos sprendimai atsižvelgia į šią problemą ypač rimtai. Užtikrinti duomenų saugumą debesų kompiuterijos aplinkoje yra daug sunkiau nei tradicinėse informacinėse sistemose [97]. Tradiciniai prieigos prie duomenų apsaugos metodai, pagrįsti tapatybės valdymu ir leidimų suteikimu, nebėra perspektyvus sprendimas apsaugoti duomenis, esančius debesijos paslaugas teikiančiose sistemose. Be to, tyrimai rodo, kad debesų kompiuterijos

sprendimai sukelia papildomą riziką, nes reikalaujama perduoti pagrindines paslaugas trečiųjų šalių paslaugų teikėjams, todėl tampa sunkiau įrodyti reikalavimų laikymąsi (angl. *compliance*), išlaikyti duomenų privatumą ir užtikrinti reikiamą paslaugų prieinamumą. Debesų kompiuterijos paslaugas naudojančios programos jau išsprendė privačių duomenų apsaugos problemas, kurias vis dar turi išspręsti viešųjų paskirstytųjų skaičiavimų programos. Tos pačios privatumo problemos galioja ir didžiųjų duomenų tyrybos užduotims. Ypač gali būti sunku užkirsti kelią nesankcionuotai prieigai prie duomenų, kurie yra paskirstyti po įvairias aplinkas (serverius, asmeninius kompiuterius, išmaniuosius įrenginius). Debesų kompiuterijos technologiją taikantys sprendimai teikia daug naudos, tačiau taip pat kelia tam tikrų iššūkių, tokių kaip: saugumo, reikalavimų laikymosi (angl. *compliance*), duomenų apsaugos ir teisinių klausimų (susijusių su duomenų perdavimu trečiosioms šalims) sprendimo. Šie iššūkiai taip pat apima kai kuriuos didesnius rūpesčius, susijusius su duomenų saugojimu, vykdomosios aplinkos ir tinklų saugumu [47]. Atsirandančios naujos technologijos sugeneruos dar daugiau duomenų, kurie galės būti naudojami didžiųjų duomenų tyrybos užduotims atlikti. Šie papildomi duomenys taip pat turės būti apsaugoti nuo neteisėtos prieigos, redagavimo ir klastojimo, paslaugų blokavimo (angl. *denial of service*) ir kitų išpuolių [71]. Daiktų debesija (angl. *cloud of things*; CoT) ir kitos panašios technologijos sudarys naujų būdų įsilaužėliams gauti prieigą prie duomenų [1]. Duomenų privatumo apsauga bus ypač svarbi hibridiniuose debesyse (privačiosios ir viešosios debesijos paslaugų derinyje), naudojamuose didžiųjų duomenų tyryboje ir kitose su duomenų apdorojimu susijusiose užduotyse. Pirmiausia reikia išspręsti saugumo problemas ir užtikrinti būtiną sąlygą sukurti patikimą aplinką skaičiavimams, pelnyti naudotojų pasitikėjimą, leidžiantį apsvarstyti galimybę naudoti šią technologiją [47, 71, 97]. Duomenų privatumo problemas debesų kompiuterijos platformose galima suskirstyti į šias kategorijas [97]:

1. prieigos prie duomenų valdymas siekiant užkirsti kelią neteisėtam duomenų perpardavimui;
2. duomenų replikacijos valdymas siekiant užkirsti kelią duomenų praradimui ir neteisėtiems pakeitimams;
3. asmens informacijos reikalavimų laikymosi priežiūra;
4. debesijos subrangovų dalyvavimo duomenų tvarkyme lygio priežiūra.

Debesų kompiuterijos modelis taip pat apima išteklių saugumo, valdymo ir stebėjimo problemas. Debesijos paslaugų naudotojų požiūriu saugumas yra didžiausias rūpestis, kuris trukdo priimti debesų kompiuterijos modelį. Įmonės ne tik praranda savo IT išteklių kontrolę perkeldamos saugumo kontrolę trečiosioms šalims, bet ir padidina išpuolių tikimybę įtraukdamos

savo duomenis į viešai prieinamą infrastruktūrą. Nors visi šie klausimai yra išspręsti, sprendimai nėra standartizuoti. Dėl debesų kompiuterijos aplinkos dinamikos šiuo metu nėra standartų ir reglamentų, apibrėžiančių programų diegimą debesijoje, tad trūksta debesų kompiuterijos standartizacijos kontrolės. Kiekvienas debesijos paslaugų teikėjas turi savo taisykles, kaip diegti programas debesų kompiuterijos aplinkoje. Debesų kompiuterija organizacijoms gali sutaupyti laiko ir pinigų, tačiau pasitikėjimas sistema yra daug svarbesnis, nes duomenys – viena iš organizacijos didžiausių vertybių, tad jų saugumas kelia didžiausią susirūpinimą. Net jei duomenų patikėjimas trečiųjų šalių platformai gali sutaupyti organizacijos laiko ir pinigų, jis kelia papildomų rizikų. Norint naudotis reikiamomis debesų kompiuterijos paslaugomis, duomenys turi būti importuojami į debesį, įkeliami tiesiai į trečiųjų šalių duomenų bazę arba informacines sistemas (pavyzdžiui, reliacinę duomenų bazę) [97]. Dėl saugumo trūkumo gali būti užkirstas kelias paskirstytųjų skaičiavimų modelio priėmimui. Įmonės privalo kaupti duomenis viešosiose infrastruktūrose ir perduoti saugos valdymą trečiosioms šalims. Šie veiksmai dar labiau sumažina IT infrastruktūros kontrolę ir padidina išpuolių tikimybę. Prisirišimas prie tiekėjo teikiamų paslaugų (angl. *vendor lock-in*), sistemų pasiekiamumas, tinklo pralaidumas, duomenų privatumas ir teisinės pasekmės yra tik kelios iš problemų, kurias nurodo debesų kompiuterijos oponentai [74].

Ši apžvalga aiškiai parodo, kad įmonės ir organizacijos yra suinteresuotos naudotis išorinėmis paslaugomis, tokiomis kaip debesų kompiuterija, jei nėra rimtos grėsmės duomenų saugumui. Tačiau net ir tada, kaip cituota [1], organizacijos negali būti tikros, kad jų duomenys yra saugūs: 2013 m. sausio 30 d. „The Independent“ paskelbė straipsnį, kuriame teigiama: „Britanijos interneto naudotojų asmeninė informacija, saugoma naudojant pagrindines debesų kompiuterijos paslaugas gali būti pasiekiama ir stebima JAV valdžios institucijų.“ Tai rodo, kad jautrūs ir privatūs duomenys neturėtų būti saugomi nepatikimoje aplinkoje. Turi būti naudojamas gerai apsaugotas serveris, esantis toje pačioje valstybėje ar patikimame geografiniame regione kaip ir pats paslaugų naudotojas [1].

2.6 Skyriaus išvados

Atliktų eksperimentinių tyrimų ir literatūros apžvalgos rezultatai leidžia daryti šias išvadas:

1. didžiųjų duomenų tyrybos procesus galima vykdyti su labai mažomis sąnaudomis, naudojant turimą IT infrastruktūrą;

2. viešųjų paskirstytųjų skaičiavimų modelis leidžia apdoroti didelius duomenų kiekius, netrikdant kitų vykstančių įmonės darbo procesų;
3. viešųjų paskirstytųjų skaičiavimų platformos kūrimą galima apibendrinti ir susieti su šių problemų sprendimu:
    a. skaičiavimuose dalyvaujantys kompiuteriai turi skirtingas architektūras;
    b. skaičiavimuose dalyvaujantys kompiuteriai yra skirtingų našumų;
    c. skaičiavimuose dalyvaujantys kompiuteriai bet kada gali tapti nepasiekiami;
    d. užduočių sąrašas ir užduočių vykdymo laikai ne visada yra iš anksto yra žinomi.
4. nėra sukurta viešųjų paskirstytųjų skaičiavimų platforma didelių duomenų apdorojimui, kurioje būtų išspręsta užduočių įstrigimo problema heterogeniškuose paskirstytųjų skaičiavimų tinkluose nenaudojant užduočių replikacijos arba informacijos apie užduočių dydžius.

3. UŽDUOČIŲ ĮSTRIGIMO PROBLEMA IR JOS SPRENDIMAS

Šiame skyriuje pateikiami disertacijos darbo rezultatai: hibridinių paskirstytųjų skaičiavimų platforma, užduočių paskirstymo algoritmo modifikacija. Be to, apžvelgiami hierarchiniai ir nehierarchiniai užduočių paskirstymo algoritmai, kurie taip pat tinkami naudoti hibridinių paskirstytųjų skaičiavimų platformose, ir sprendžiama užduočių įstrigimo problema.

Užduočių įstrigimo paskirstytųjų skaičiavimų tinkluose problema – viena pagrindinių priežasčių, neleidžiančių išnaudoti įmonių vidinių skaičiavimo išteklių. Užduotys lygiagrečiai paskirstomos tarp turimų skaičiavimo išteklių, tačiau taip vykdant užduotis hibridiniuose skaičiavimo tinkluose pasitaiko užduočių įstrigimo problema [43]. Užduotys vadinamos įstrigusiomis, kai yra vykdomos lėčiau už vidutinę užduoties atlikimo trukmę [76]. Neįprastai lėta užduotis paprastai identifikuojama kaip bet kokia užduotis, kurios užduoties atlikimo laikas yra 50 % ilgesnis už vidutinį užduoties atlikimo laiką darbo etape [77, 83]. Užduočių įstrigimo problemą gali sukelti daug veiksnių (apžvelgiami 3.1 skyriuje). Remiantis [39], veiksniai gali tarpusavyje koreliuoti.

3.1 Užduočių įstrigimo priežastys

Pagrindinė užduočių įstrigimo priežastis – užduočių varžymasis dėl laisvų išteklių. Ši priežastis tampa ypač aktuali, kai ištekliai yra heterogeniniai. Tačiau užduotys taip pat gali įstrigti dėl išteklių gedimo ir kitų priežasčių. Remiantis [43], 8 priežastys lemia įstrigusių užduočių atsiradimą:

1. Duomenų surinkimas – užduotys gali įstrigti dėl informacijos užlaikymo skirtinguose sistemos lygiuose.
2. Procesoriaus naudojimas – nustatyta, kad yra didelė koreliacija tarp intensyvaus sistemos procesoriaus naudojimo ir įstrigusių užduočių atsiradimo. Šio įvykio priežastis – varžymasis dėl išteklių, kuris gali kilti dėl lygiagrečiai vykdomų procesų.
3. Planavimas – nustatyta, kad planavimas ir išteklių paskirstymo sprendimai taip pat lemia įstrigusių užduočių atsiradimą. Dėl prastų užduočių paskirstymo mechanizmų ištekliai išeikvojami ir užduočių vykdymas sulėtėja. QoS reikalavimų dinamiškumas vykdymo metu lemia nesugebėjimą efektyviai valdyti išteklių, o tai lemia dar didesnį įstrigusių užduočių skaičių. Atliekant išteklių planavimą, įstrigusios užduotys gali atsirasti šiose situacijose:

a. neefektyviai planuojami ištekliai užduočių vykdymui, todėl ištekliai neefektyviai paskirstomi užduotims neatlikus išteklių optimizavimo;
b. nenaudojami ištekliai yra aktyvūs, todėl sunaudoja daugiau energijos ir daro įtaką kitų išteklių efektyvumui (kai kuriems ištekliams reikia daugiau energijos, kad jie galėtų nuolat veikti).

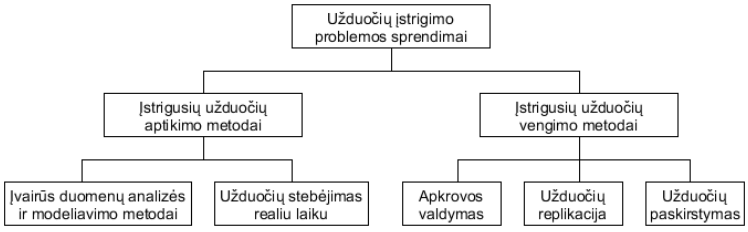
4. Neprieinamas vietinis diskas – užduotys gali įstrigti, kai laukiama prieigos prie kompiuterio standžiojo disko. Tokį nepasiekiamumą daugiausia lemia:
    a. didėjantis atsarginių (replikuotų) užduočių kiekis;
    b. gautų rezultatų neišsaugojimas. Kartais saugant išvestį diske gali įvykti klaida, kelianti trikdžių, kai kitos užduotys nori pasiekti tuos duomenis vykdymo metu.
5. Blogas įvesties duomenų paskirstymas – užduočių įstrigimą lemia skirtingi duomenų dydžiai ir laiko skirtumai gaunant įvesties duomenis. Jei bendras duomenų rinkinys yra netolygiai paskirstomas tarp užduočių, kai kurios užduotys gali užtrukti žymiai ilgiau, lyginant su kitomis to paties tipo užduotimis. Netolygus duomenų paskirstymas taip pat gali turėti įtakos duomenų prieigai ir duomenų apdorojimui, tiesiogiai lemdamas delsą tarp užduočių ir dar labiau didindamas užduočių įstrigimo tikimybę.
6. Varžymasis dėl išteklių – užduotys gali įstrigti įvykus konfliktui dėl prieigos prie skaičiavimo išteklių. Konfliktai gali kilti dėl šių priežasčių:
    a. Išteklių heterogeniškumas – pagrindinė varžymosi dėl išteklių priežastis, atsirandanti dėl aparatūros skirtumų, dėl kurių blogėja užduočių vykdymas;
    b. Prastas užduočių paskirstymo algoritmas – algoritmai planuoja išteklius neveiksmingai, o tai gali padidinti išteklių sunaudojimą ir lemti reikalingų išteklių nepasiekiamumą;
    c. Užduočių replikavimas – replikuojant užduotis kuriamos jų kopijos, tikintis, kad jos bus įvykdytos greičiau naudojant kitus skaičiavimo išteklius. Šis sprendimas naudoja žymiai daugiau išteklių ir gali sukelti užduočių įstrigimo problemą;
    d. Neveiksmingas išteklių planavimo metodas – gali lemti neefektyvų išteklių paskirstymą ir didinti išteklių naudojimą, o tai savo ruožtu gali sukelti užduočių įstrigimo problemą;
    e. Laikini sistemos sulėtėjimai – gali atsirasti dėl neefektyvaus išteklių paskirstymo;

f. Išteklių poreikis viršija turimų išteklių kiekį.

7. Užduoties vykdymas – vykdant užduotį gali kilti kliūčių dėl neapdorotų įvesties duomenų, kuriuos turi apdoroti kitos užduotys. Laukiančios užduotys gali įstrigti. Be to, užduočių nesuderinamumas dėl skirtingų darbo krūvių ar reikalavimų gali lemti sistemos sulėtėjimą.
8. Programinės ir aparatinės įrangos gedimai – užduotys gali įstrigti dėl įvairių gedimų. Paminėtina, kad tokių gedimų toleravimas ir atkūrimo mechanizmai taip pat gali sukelti užduočių įstrigimą (pavyzdžiui, sukuriant naujų konfliktų dėl papildomų išteklių).

3.2 Užduočių įstrigimo problemos sprendimai

Užduočių įstrigimo problemos sprendimo būdus galima suskirstyti į dvi kategorijas: įstrigusių užduočių aptikimo ir įstrigusių užduočių vengimo metodus [43].



12 pav. Užduočių įstrigimo problemos sprendimo metodai [43]

Remiantis [43], įstrigusių užduočių aptikimo metodai apima įvairius modeliavimo, istorinių duomenų analizės ir sistemos stebėjimo realiuoju laiku metodus. Šie metodai nėra tinkami, kai apie vykdomas užduotis nėra jokios informacijos. Užduočių replikacija savaime gali sukelti įstrigimo problemą heterogeniškuose skaičiavimo tinkluose. Užduočių replikacijos ir apkrovos valdymo metodai taip pat reikalauja informacijos apie vidutinę užduoties vykdymo trukmę. Todėl toliau nagrinėjami užduočių paskirstymo metodai, kuriais siekiama išvengti užduočių įstrigimo problemos.

3.3 Užduočių paskirstymo algoritmai

Šiame skyriuje apžvelgiami egzistuojantys hierarchiniai ir nehierarchiniai užduočių paskirstymo algoritmai, kurie galėtų būti tinkami aukščiausio lygio užduočių planuokliui. Algoritmas turi būti suderinamas su siūloma hibridinių

paskirstytųjų skaičiavimų platformos architektūra, taigi bus apžvelgti egzistuojantys šiai klasifikacijai priskiriami hierarchiniai užduočių paskirstymo algoritmai [34]:

- globalūs – užduotys vykdomos keliuose skaičiavimo mazguose naudojant kelis klasterius;
- dinamiški – užduočių srautas yra dinamiškas, o užduočių vykdymo kaštai nežinomi;
- fiziškai paskirstyti – planavimas atliekamas naudojant kelis paskirstytus planuoklius;
- bendradarbiaujantys – visi užduočių planuokliai bendradarbiauja siekdami priimti geresnius planavimo sprendimus.

Taip pat 3.3.3 skyriuje pristatytas patobulintas apkrovos balansavimo metodas FIFO, užduotims paskirstyti naudojantis užduočių sulaikymo buferį. Nors šis metodas naudojamas paskirstyti užduotis eilių sistemose (angl. *queueing systems*) su dviem heterogeniškais serveriais, šis metodas bus pritaikytas ir įdiegtas skaičiavimams, atliekamiems paskirstant užduotis tarp dviejų ir daugiau klasterių.

3.3.1 Hierarchiniai užduočių paskirstymo algoritmai

Egzistuojančius hierarchinius planavimo algoritmus [34], skirstančius užduotis į kelis klasterius, galima apibendrinti naudojant šiuos keturis esamus sprendimo būdus, kur:

- užduotys perkeliamos iš labai apkrautų klasterių į mažiau apkrautus kaimyninius klasterius, darant prielaidą, kad užduočių patekimo į sistemą srautas neviršys paslaugų išteklių galimybių [75];
- užduotys išskaidomos į mažesnes (angl. *subtasks*), o laiko apskaičiavimas atliekamas naudojant modeliavimo rezultatus, įskaitant galimus išteklių paskirstymo konfliktus. Mažesnės užduotys paskirstomos turimiems vietiniams klasteriams, kurie gali įvykdyti užduotis greičiausiai [56];
- užduotys rūšiuojamos mažėjančia tvarka pagal vidutinę vykdymo trukmę ir paskirstomos turimiems skaičiavimo ištekliams. Vykdymo trukmė apskaičiuojama naudojant modeliavimo rezultatus kartu su istoriniais duomenimis [48];
- lėto veikimo užduotys replikuojamos tikintis rezultatus gauti greičiau iš kito tuos pačius skaičiavimus atliekančio išteklio [81].

Esami sprendimai arba daro prielaidą, kad užduočių srautas ir visos sistemos aptarnavimo pajėgumai visada išlieka stabilūs, veikia numatytu

užduoties vykdymo laiku arba naudoja užduočių replikaciją. Siūlomas užduočių paskirstymo metodas, naudojantis sulaikymo buferį, skiriasi nuo šiuo metu egzistuojančių sprendimų, nes yra skirtas veikti heterogeniškoje aplinkoje be jokių modeliavimo rezultatų ar užduočių replikavimo.

3.3.2 Paskirstytųjų skaičiavimų užduočių planuokliai

Šiame skyriuje apžvelgiami plačiai naudojami, tarpusavyje nepriklausomų darbų planavimo algoritmai. Kai kurie iš jų taip pat naudojami didelių duomenų tyrybos užduotims „Facebook“, „Yahoo“, „Hadoop“. Yra mažiausiai penki gerai žinomi užduočių planavimo algoritmai [40, 54, 72, 109]:

- „Fair-share“ [40, 72] – kiekvienam darbui skiriama vienoda išteklių dalis;
- FIFO [40, 72] – seniausios užduotys vykdomos mazgų, kurie atsilaisvina pirmiausiai;
- „Capacity“ [40, 72] – ištekliai paskirstomi darbų apdorojimo eilėms (angl. *processing queues*);
- LATE [40, 54, 72] – replikuoja užduotis, įstrigusias lėtuose skaičiavimo mazguose, naudodamas tokias replikuotas užduotis kaip atsarginę kopiją (patikimumas negarantuojamas [72]);
- „Round-robin“ [109] – vykdo visas programas nuo pirmosios užduoties pirmame mazge, visas programas iš antros užduoties antrame mazge ir pan.

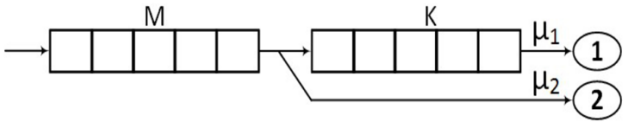
Vienintelis algoritmas, galintis paskirstyti gaunamą dinaminių užduočių srautą labai heterogeniškoje aplinkoje į du klasterius, yra FIFO. Kiti gerai žinomi užduočių planavimo algoritmai, tokie kaip „Min-min“ [15, 34], „Min-max“ [15, 34], MCT [34, 109], „Suffrage“ algoritmas [15], nepritaikomi, nes šiems algoritmams iš anksto reikalingas visų užduočių ir skaičiavimo išteklių sąrašas. Algoritmai, tokie kaip „User Defined Assignment“ [15], taip pat netinka, nes užduotys griežta tvarka priskiriamos ištekliams, kuriuose, manoma, užduotys bus įvykdytos greičiausiai, nepriklausomai nuo to, ar tie ištekliai prieinami.

3.3.3 Užduočių sulaikymo buferis

Remiantis [61], užduočių sulaikymo buferis (žr. 13 pav.) pagerina užduočių sąrašo įvykdymo trukmę eilių sistemose (angl. *queuing systems*) su dviem heterogeniškais serveriais. Užduočių sulaikymo buferis sumažina lėtojo serverio apkrovą nukreipdamas daugiau užduočių į greitąjį serverį. Jei

greitasis serveris užimtas, sulaikymo buferiui pridedamos naujos užduotys. Jei buferis užpildytas, užduotį gauna lėtasis serveris.

Šį metodą galima pritaikyti siekiant pagerinti užduočių paskirstymą tarp dviejų skirtingo našumo paskirstytųjų skaičiavimų klasterių. Galima daryti prielaidą, kad privatusis klasteris visada užduotis atliks greičiau už viešąjį klasterį. Todėl galima naudoti užduočių sulaikymo buferį siekiant sumažinti viešajam klasteriui paskiriamų užduočių skaičių. Tokiu būdu sumažinama užduočių atlikimo trukmė ir pagerinamas platformos teikiamų paslaugų patikimumas.



13 pav. Eilių sudarymo sistemos (angl. queuing system) su užduočių sulaikymo buferiu schema [61]

Čia M yra buferio dydis, skirtas naujoms užduotims laikyti, K yra užduočių sulaikymo buferio dydis, μ_1 – greitojo kanalo efektyvumas, μ_2 – lėtojo kanalo efektyvumas, 1 – greitasis kanalas, 2 – lėtasis kanalas. Tada remiantis [61], užduoties sulaikymo buferio ilgis K gali būti išreiškiamas taip:

$$K \approx r \cdot (1 - q)\,, \tag{2}$$

kur r yra greitojo ir lėtojo kanalų efektyvumų santykis, o q – užduočių vykdymo efektyvumo koeficientas:

$$r = \frac{\mu_1}{\mu_2}, \tag{3}$$

$$q = \frac{c}{t \cdot m \cdot \mu_1}\,, \tag{4}$$

kur c – atliktų užduočių skaičius, t – užduočių vykdymo laikų suma, m – greitojo kanalo mazgų skaičius, o μ_1 – greitojo kanalo efektyvumas:

$$\mu_1 = \frac{a_1}{b_1}\,, \tag{5}$$

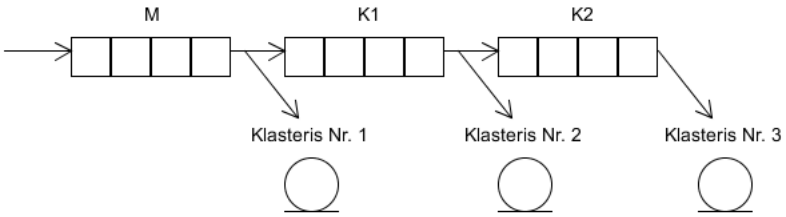
kur a_1 – atliktų užduočių skaičius naudojant greitąjį kanalą, o b_1 – laikas, reikalingas toms užduotims atlikti. μ_2 – lėtojo kanalo efektyvumas:

$$\mu_2 = \frac{a_2}{b_2}, \tag{6}$$

kur a_2 – atliktų užduočių skaičius naudojant lėtąjį kanalą, o b_2 – laikas, reikalingas toms užduotims atlikti.

3.3.4 Užduočių sulaikymo buferis daugiakanalėse sistemose

Užduočių sulaikymo buferis daugiakanalėse sistemose gali būti pritaikomas labai panašiu būdu. Remiantis [61], užduočių sulaikymo buferis labiausiai sutrumpina užduočių atlikimo laiką, kai efektyvumų skirtumas tarp dviejų užduočių apdorojimo sistemų yra didžiausias. Todėl paskirstytųjų skaičiavimų platformos, turinčios n klasterių ($n > 1$), atveju, užduotys turi būti skirstomos tarp lėčiausio klasterio (lėtojo kanalo) ir likusių $n - 1$ klasterių (greitojo kanalo) [59]. Užduočių sulaikymo buferio taikymo pavyzdys trijų kanalų sistemoje pateiktas 14 pav.



14 pav. Užduočių paskirstymo buferis trijų kanalų sistemoje

Kaip ir dviejų kanalų sistemoje, M – laukimo buferio ilgis, o $K1$ ir $K2$ – užduočių sulaikymo buferių ilgiai. Klasteriai atitinka skirtingų našumų kanalus, kur „*Klasteris Nr. 1*“ yra mažiausio, o „*Klasteris Nr. 3*“ yra didžiausio efektyvumo kanalai. Užduotys iš pradžių patenka į didžiausio efektyvumo klasterį („*Klasteris Nr. 3*“). Jei šis klasteris užimtas, užduotys patenka į užduočių laukimo buferį, kurio ilgis $K2$. Jei užduočių sulaikymo buferis pilnas, užduotys patenka į antrą pagal efektyvumą klasterį („*Klasteris Nr. 2*“).

3.4 Algoritmai

Šiame skyriuje pristatomi algoritmai, skirti užduočių sulaikymo buferio ilgiui paskaičiuoti ir užduotims paskirstyti į BOINC ir „Apache Mesos“ klasterius. Šie algoritmai naudojami sukurtoje hibridinių paskirstytųjų skaičiavimų platformoje, tad tolesniuose (3.4.1, 3.4.2, 3.4.3) skyriuose

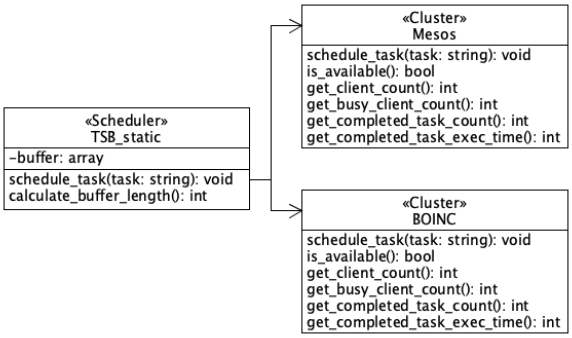
pateikiamas algoritmų pseudokodas aiškiai parodo, kaip veikia platforma ir kaip renkami reikiami duomenys užduočių sulaikymo buferio ilgiui gauti.

Užduočių sulaikymo buferio ilgiui įvertinti reikalingos šios funkcijos (žr. 15 pav.):

- *is_available* – nurodo, ar klasteris turi laisvų išteklių, galinčių priimti naują užduotį;
- *get_client_count* – grąžina klasteryje esančių klientų kiekį;
- *get_busy_client_count* – grąžina klasteryje esančių klientų kiekį, kuriems paskirtos vykdyti užduotys;
- *get_completed_task_count* – grąžina klasteryje įvykdytų užduočių kiekį;
- *get_completed_task_exec_time* – grąžina laiką, kurio klasteriui prireikė įvykdyti užduotis.

Visos trys klasės turi *schedule_task* metodą, skirtą užduočių paskirstymui. Užduočių skirstytuvas šį metodą naudoja užduoties priskyrimui vienam iš klasterių, o klasteriai naudoja šį metodą užduoties priskyrimui klientams.

Toliau apžvelgiamas šių funkcijų įgyvendinimas kiekvieno klasterio atveju.



15 pav. Klasių diagrama

3.4.1 BOINC klasteriui skirtų algoritmų realizacija

BOINC klasteris visą reikiamą informaciją saugo MySQL duomenų bazėje, tad reikiamus duomenis užduočių sulaikymo buferio ilgiui paskaičiuoti galima gauti panaudojus „host“ ir „result“ lentelėse esančius duomenis. „Host“ lentelėje saugomi visų skaičiavimo išteklių (agentų) duomenys, o „result“ lentelėje – užduočių ir jų gautų rezultatų duomenys. Toliau pateikti algoritmai (žr. 16–20 pav.) aprašo, kaip šią informaciją galima gauti ir panaudoti.

Algoritmas Nr. 1: get_client_count
Įvestis: nėra
initialize count **to EXECUTE** (“**SELECT COUNT**(*) **FROM** `host`”)
Išvestis: count

16 pav. Algoritmas, grąžinantis agentų kiekį BOINC klasteryje

Algoritmas Nr. 2: get_completed_task_count
Įvestis: nėra
initialize count **to EXECUTE** (“**SELECT COUNT**(*) **FROM** `result` **WHERE** `outcome` = 1”)
Išvestis: count

17 pav. Algoritmas, grąžinantis BOINC klasteryje atliktų užduočių kiekį

Algoritmas Nr. 3: get_completed_task_exec_time
Įvestis: nėra
initialize time **to EXECUTE** (“ **SELECT ROUND**(**SUM**(`elapsed_time`)) **FROM** `result` **WHERE** `outcome` = 1”)
Išvestis: time

18 pav. Algoritmas, grąžinantis laiką, kurio prireikė BOINC klasteryje atlikti užduotis

Užduotis gavusių klientų kiekio įvertinimas – sudėtingesnė užduotis (žr. 19 pav.), nes BOINC klasterio užduočių paskirstymo algoritmas gali nuspręsti, kiek užduočių priskirti vienam skaičiavimo ištekliui vienu metu. Siekiant to išvengti, BOINC klasterį galima sukonfigūruoti taip, kad vienam skaičiavimo ištekliui nebūtų leidžiama vykdyti kelių užduočių vienu metu pasinaudojant *max_jobs_in_progress* parametru *config_aux.xml* byloje. Tačiau jei naujos užduotys BOINC klasteriui pateikiamos dažniau nei įvykdomos turimos užduotys, vienam skaičiavimo ištekliui bus priskiriamos kelios užduotys. Todėl vertinant užimtų klientų kiekį reikia įvertinti ir jau priskirtų, tačiau savo vykdymo eilės vis dar laukiančių, užduočių kiekį.

Algoritmas Nr. 4: get_busy_client_count

Įvestis: nėra

```
initialize client_count to call: get_client_count
initialize busy_client_count to zero

initialize uncompleted_task_count_by_client to EXECUTE (“
   SELECT `hostid`, COUNT(*) FROM `result`
   WHERE `outcome` = 0 GROUP BY `hostid`”)

for each (client_id, task_count) in uncompleted_task_count_by_client do
    if client_id is equal to zero
       add task_count to busy_client_count
    else
       add 1 to busy_client_count
    endif
endfor

if busy_client_count is greater than client_count
   set busy_client_count to client_count
endif
```

Išvestis: busy_client_count

19 pav. Algoritmas, grąžinantis klientų kiekį BOINC klasteryje, kuriems yra priskirtos vykdyti užduotys

Algoritmas Nr. 5: is_available

Įvestis: nėra

```
initialize client_count to call: get_client_count
initialize busy_client_count to call: get_busy_client_count
initialize is_available to false

if busy_client_count is less than client_count
   set is_available to true
endif
```

Išvestis: is_available

20 pav. Algoritmas, nurodantis, ar BOINC klasteris turi laisvų išteklių, galinčių priimti naują užduotį

Naujoms užduotims kurti BOINC klasteryje galima panaudoti „Docker“ konteineryje (pavadinimu „boinc-server_apache_1“) esančią vykdomąją bylą „bin/boinc2docker_create_work.py” (žr. 21 pav.).

Algoritmas Nr. 6: schedule task
Įvestis: task
call: docker exec boinc-server_apache_1 bin/boinc2docker_create_work.py (**arguments:** task)
Išvestis: nėra

21 pav. Algoritmas, skirtas BOINC klasteryje kurti naujas užduotis

3.4.2 „Apache Mesos“ klasteriui skirtų algoritmų realizacija

„Apache Mesos“ klasteris visą reikiamą informaciją saugo „Apache Zookeeper“ duomenų bazėje, iš kurios duomenis galima gauti naudojant „Apache Mesos“ REST API (pseudokode šis veiksmas atliekamas naudojant *send_message* funkciją). Toliau pateikti algoritmai naudojami užduočių sulaikymo buferio ilgiui paskaičiuoti (žr. 22–29 pav.).

Algoritmas Nr. 7: get_client_count
Įvestis: nėra
initialize url **to** "http://*master.server.domain*:5050/master/state" **initialize** state **to call:** send_message (**arguments:** url) **initialize** count **to** state["activated_slaves"]
Išvestis: count

22 pav. Algoritmas, grąžinantis klientų kiekį „Apache Mesos“ klasteryje

Siekiant neapkrauti serverio užklausomis apie vykdomų užduočių būsenas, toliau pateikiami du papildomi algoritmai *get_existing_tasks* ir *refresh_active_tasks*. Šie algoritmai sukuria ir periodiškai atnaujina vykdomų užduočių masyvą. Taip, perskaičiuojant užduočių sulaikymo buferio ilgį, užuot kaskart kreipusis į „Zookeeper“ duomenų bazę, galima naudotis turimu užduočių masyvu, apdorojant informaciją apie visas iki tol įvykdytas ir tam tikru momentu vykdomas užduotis.

Algoritmas Nr. 8: get_existing_tasks

Įvestis: nėra

```
initialize work_cnt to zero;
initialize task_id to zero;
initialize active_tasks to empty array
initialize url to "http://master.server.domain:8080/v1/scheduler/jobs/"
initialize job_status_list to call: send_message (arguments: url)

for each task in job_status_list do
   if task["lastSuccess"] is not equal to zero
      add task["name"] to active_tasks
   end if

   set task_id to task["name"] without first 9 characters

   if task_id is equal or greater than work_cnt
      set work_cnt to task_id + 1
   endif
end for
```

Išvestis: work_cnt, active_tasks

23 pav. Algoritmas, skirtas atnaujinti „Apache Mesos“ klasteryje atliktų ir dar vykdomų užduočių būsenas

Algoritmas Nr. 9: refresh_active_tasks

Įvestis: active_tasks

```
initialize url to empty string
initialize job_status to empty array

for each task in active_tasks do
   set url to "http://master.server.domain:8080/v1/scheduler/jobs/search?
   name=" & task
   set task_status to call: send_message (arguments: url)

   if task_status["lastSuccess"] is not equal to zero
      remove task from active_tasks
   end if
end for
```

Išvestis: active_tasks

24 pav. Algoritmas, skirtas atnaujinti „Apache Mesos“ klasteryje vykdomų užduočių būsenas

Algoritmas Nr. 10: get_completed_task_count
Įvestis: nėra

initialize count **to** zero
initialize url **to** "http://*master.server.domain*:5050/master/state"
initialize stats **to call:** send_message (**arguments:** url)
 	
for each task **in** stats["frameworks"0]["completed_tasks"] **do**
  for each task_status **in** task["statuses"] **do**
  if task_status["state"] **is equal to** "TASK_FINISHED"
  and task_status["timestamp"] **is greater than** zero **then**
  add 1 **to** count
  end if
  end for
end for

Išvestis: count

25 pav. Algoritmas, grąžinantis „Apache Mesos“ klasteryje atliktų užduočių kiekį

Algoritmas Nr. 11: get_completed_task_exec_time
Įvestis: nėra

initialize time **to** zero
initialize task_execution_time **to** zero
initialize task_status_map **to** empty array
initialize url **to** "http://*master.server.domain*:5050/master/state"
initialize stats **to call:** send_message (**arguments:** url)
 	
for each task **in** stats["frameworks"][0]["completed_tasks"] **do**
  for each task_status **in** task["statuses"] **do**
  set task_status_map[task_status["state"]] **to** task_status["timestamp"]
  end for
 	
  if task_status_map["TASK_FINISHED"] **is greater than** zero

```
        set task_execution_time to task_status_map["TASK_FINISHED"] -
          task_status_map["TASK_STARTING "]

        add task_execution_time to time
      end if

      set task_status_map to empty array
end for
```

Išvestis: time

26 pav. Algoritmas, grąžinantis laiką, kurio prireikė „Apache Mesos“ klasteryje atlikti užduotis

Pasinaudojus *get_existing_tasks* algoritmu, gaunamas aktyvių užduočių masyvas. Kadangi vienam skaičiavimo ištekliui vienu metu priskiriama tik viena užduotis, užimtų išteklių kiekį galima paskaičiuoti pagal tam tikru metu aktyvių užduočių skaičių.

Algoritmas Nr. 12: get_busy_client_count

Įvestis: active_tasks

```
initialize busy_client_count to length(active_tasks)
```

Išvestis: busy_client_count

27 pav. Algoritmas, grąžinantis klientų kiekį „Apache Mesos“ klasteryje, kuriems yra priskirtos vykdyti užduotys

Algoritmas Nr. 13: is_available

Įvestis: nėra

```
initialize client_count to call: get_client_count
initialize busy_client_count to call: get_busy_client_count
initialize is_available to false

if busy_client_count is less than client_count
   set is_available to true
endif
```

Išvestis: is_available

28 pav. Algoritmas, nurodantis, ar „Apache Mesos“ klasteris turi laisvų išteklių, galinčių priimti naują užduotį

Naujoms užduotims kurti „Apache Mesos“ klasteryje siunčiame POST užklausą į „Apache Mesos“ REST API.

Algoritmas Nr. 14: schedule_task

Įvestis: task

```
initialize url to "http://master.server.domain:8080/v1/scheduler/iso8601"
initialize task_name to "cpp-test-" & work_cnt
initialize current_time to "%Y-%m-%dT%H:%M:%S.000+03:00"
initialize data to "
{
    schedule: ‘R1 current_time P10M’,
    name: task_name,
    cpus: 1,
    mem: 128,
    retries: 0,
    command: ‘docker run task’
}"

add task_name to active_tasks

call: send_message (arguments: url, data)
```

Išvestis: nėra

29 pav. Algoritmas, skirtas „Apache Mesos“ klasteryje kurti naujas užduotis

3.4.3 Užduočių paskirstymo algoritmai

Šiame skyriuje pateikiami užduočių paskirstymo algoritmai, naudojantys FIFO ir užduočių sulaikymo buferio metodus (žr. 30–32 pav.). Šie algoritmai naudoja ankstesniuose skyriuose aprašytus algoritmus.

Algoritmas Nr. 15: distribute_task_using_FIFO

Įvestis: task

```
initialize target_found to false
initialize mesos_is_available to false
initialize boinc_is_available to false
```

```
do
   set mesos_is_available to call: mesos.is_available
   set boinc_is_available to call: boinc.is_available

   if mesos_is_available is equal to true
      call: mesos.schedule_task (arguments: task)
      set target_found to true
    else if boinc_is_available is equal to true
       call: boinc.schedule_task (arguments: task)
       set target_found to true
    else
       Wait 1 second
    end if
while target_found is equal to false
```

Išvestis: nėra

30 pav. FIFO užduočių paskirstymo algoritmas

Algoritmas Nr. 16: calculate_buffer_length

Įvestis: nėra

```
initialize k to zero
initialize lambda_stat to zero
initialize m to call: mesos.get_client_count
initialize boinc_completed_task_exec_time to
   call: boinc.get_task_exec_time
initialize mesos_completed_task_exec_time to
   call: mesos.get_task_exec_time
initialize boinc_completed_task_count to
   call: boinc.get_completed_task_count
initialize mesos_completed_task_count to
   call: mesos.get_completed_task_count

initialize completed_task_exec_time to boinc_completed_task_exec_time
   + mesos_completed_task_exec_time
initialize completed_task_count to boinc_completed_task_count +
   mesos_completed_task_count
```

```
initialize M1 to zero
initialize M2 to zero

if boinc_completed_task_count is not equal to zero and
   mesos_completed_task_count is not equal to zero
      set M1 to mesos_completed_task_count devided by
      mesos_completed_task_exec_time
      set M2 to boinc_completed_task_count devided by
      boinc_completed_task_exec_time
      set lambda_stat to completed_task_count devided by
      completed_task_exec_time

      set k to (M1 / M2) * (1 - (lambda_stat / (m * M1)))
end if
```

Išvestis: round(k)

31 pav. Užduočių sulaikymo buferio ilgio skaičiavimo algoritmas

Algoritmas Nr. 17: distribute_task_using_task_stalling_buffer

Įvestis: task, task_buffer

```
initialize target_found to false
initialize mesos_is_available to false
initialize boinc_is_available to false
initialize k to call: calculate_buffer_length

do
   set mesos_is_available to call: mesos.is_available
   set boinc_is_available to call: boinc.is_available

   if mesos_is_available is equal to true
      call: mesos.schedule_task (arguments: task)
      set target_found to true
    else if length(task_buffer) is less than k
      add task to task_buffer
      set target_found to true
    else if boinc_is_available is equal to true
      call: boinc.schedule_task (arguments: task)
      set target_found to true
```

```
    else
      Wait 1 second
    end if
while target_found is equal to false
```

Išvestis: nėra

32 pav. Užduočių paskirstymo algoritmas, naudojantis užduočių sulaikymo buferį

3.5 Hibridinių paskirstytųjų skaičiavimų platforma

Šiame skyriuje pristatoma sukurta hibridinių paskirstytųjų skaičiavimų platforma: platformos architektūra ir duomenų formatas užduotims teikti. Platformoje taikomas 3.3.3 skyriuje pristatytas užduočių paskirstymo metodas (užduočių sulaikymo buferis), kuriam įgyvendinti panaudoti 3.4 skyriuje pristatyti algoritmai. Todėl sukurta platforma minimizuoja užduočių aibės įvykdymo trukmę heterogeniškuose hibridinių paskirstytųjų skaičiavimų tinkluose.

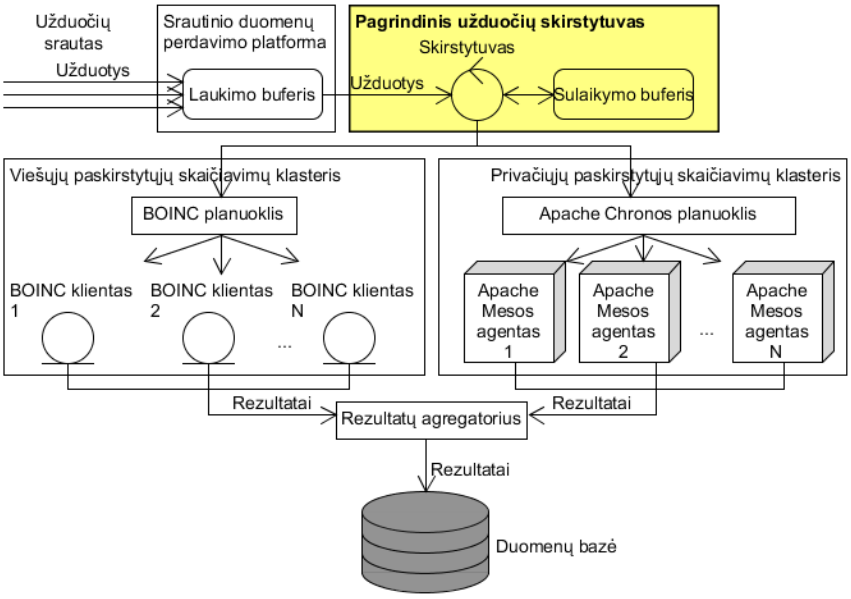
3.5.1 Architektūra

Šiame skyriuje pateikiama hibridinių paskirstytųjų skaičiavimų modelio pagrindu sukurtos platformos architektūra, leidžianti vykdyti įvairaus pobūdžio užduotis naudojant vidinius serverius (arba debesų kompiuterijos paslaugas) ir asmeninius kompiuterius. Pateiktas architektūros pavyzdys (žr. 33 pav.) tinkamas iliustruoti ir kitoms hibridinių paskirstytųjų skaičiavimų platformoms. Pagrindinis skirtumas tarp siūlomos ir kitų platformų – užduočių paskirstymo metodas (iliustracijoje pažymėtas geltonai). Siūloma platforma sujungia viešojo ir privačiojo skaičiavimo klasterius į hibridinių paskirstytųjų skaičiavimų tinklą. Ši platforma naudoja užduočių paskirstymo metodą, galintį valdyti darbo krūvį tarp dviejų klasterių be jokios papildomos informacijos apie užduotis. Paslaugų patikimumo ir užduočių įstrigimo problemos sprendžiamos naudojant siūlomą užduočių paskirstymo metodą, paremtą užduočių sulaikymo buferiu. Aprašoma platforma suteikia įmonėms galimybę sumažinti paslaugų kaštus ir vis tiek išlaikyti paslaugų patikimumą.

Aprašomoje platformoje naudojamos atvirojo kodo ir tarpusavyje suderinamos technologijos (visos palaiko tą patį programinės įrangos virtualizacijos sprendimą). Tačiau labai svarbu pažymėti, kad gali būti naudojamos ir kitos suderinamos alternatyvos.

Kaip parodyta 33 pav., siūloma hibridinių paskirstytųjų skaičiavimų platformos architektūra turi dviejų lygių hierarchiją, kurioje fiziškai

paskirstyti (hierarchiniai) bendradarbiaujantys užduočių planuokliai. Aukščiausiame lygyje yra pagrindinis planuoklis, kuris paskirsto užduotis tarp žemesnio lygio klasterių. Ši architektūra sukuria dinamišką ir lankstų pagrindą užduotims vykdyti ir suteikia daugiau galimybių valdyti paslaugų kokybę. Siekiant paskirstyti užduotis tarp įmonės serverių ir darbuotojų kompiuterių naudojami du klasteriai: privatusis (valdomas „Apache Mesos“) ir viešasis (valdomas BOINC). Kiekvieną klasterį valdo planuoklis, specialiai sukurtas konkrečiai klasterio aplinkai.



33 pav. Architektūra, sujungianti du klasterius naudojant dviejų lygių planuoklių hierarchiją

Remiantis šios sistemos projektavimo metodologija, užduočių planavimas perkeliamas į žemesnio lygio klasterius – parenkama, kuris klasteris turėtų gauti užduotį. Siūlomą architektūrą sudaro šie pagrindiniai komponentai: pagrindinis planuoklis, privačiojo skaičiavimo klasteris ir viešojo skaičiavimo klasteris. Pateiktoje architektūroje yra papildomų komponentų, kurie nėra būtini, tačiau padeda iliustruoti visą sprendimą:

- srautinio duomenų perdavimo platforma (angl. *streaming platform*) – saugo visas naujas gaunamas užduotis, kol sistema priima užduotis;
- rezultatų agregatorius (angl. *result aggregator*) – kaupia ir agreguoja atliktų užduočių rezultatus;
- duomenų bazė – saugo agreguotus rezultatus.

Pagrindinis užduočių planuoklis – pagrindinis disertacijos tyrimų objektas. 33 pav. pateiktas šiai architektūrai siūlomas užduočių planavimo metodas (pristatytas 3.3.3 skyriuje). Tai yra aukščiausiame hierarchijos lygyje esantis užduočių planuoklis, paskirstantis užduotis žemesnio lygio planuokliams. Jį sudaro šie subkomponentai:

- sulaikymo buferis – sulaiko užduotis vėlesniam apdorojimui naudojant privačiųjų skaičiavimų klasterį;
- skirstytuvas – užduočių paskirstymo algoritmas, atsakingas už užduočių paskirstymą tarp klasterių:
    - persiunčia naujas gaunamas užduotis į užduočių sulaikymo buferį, jei užduočių sulaikymo buferis nėra pilnas;
    - persiunčia naujas gaunamas užduotis BOINC užduočių planuokliui, jei sulaikymo buferis yra pilnas ir viešųjų skaičiavimų klasteryje yra laisvų BOINC klientų;
    - perduoda užduotis iš sulaikymo buferio į „Apache Mesos“ užduočių planuoklį, jei privačiųjų skaičiavimų klasteryje yra laisvų „Apache Mesos“ agentų.

Privačiųjų ir viešųjų skaičiavimų klasterius sudaro klasterių planuokliai ir klientai (arba agentai), atsakingi už kiekvieno klasterio užduočių paskirstymą ir vykdymą.

Pažymėtina, kad kai kuriais atvejais aprašoma platforma gali būti pritaikoma tam tikroms duomenų privatumo ir prieinamumo užduotims spręsti, pvz., atsisiųsti ar įkelti didelius duomenų kiekius, apdoroti privačią informaciją. Duomenų dydžio ir privatumo problemos yra gerai žinomos, be to, jau yra įvairių tokių problemų sprendimų [22, 95, 98]. Tokius sprendimus būtų galima integruoti į siūlomą platformą siekiant pagerinti duomenų saugumą ir prieinamumą. Tačiau tam reikėtų papildomų tyrimų.

3.5.2 Duomenų formatas

Šiame skyriuje apibrėžiamas duomenų formatas, skirtas teikti užduotis siūlomai hibridinių paskirstytųjų skaičiavimų platformai. Bet kuris pageidaujamas duomenų formatas tinka apibrėžti užduotis. Šiuo atveju pasirenkamas plačiai naudojamas ir lengvai skaitomas JSON formatas. Čia užduoties apibrėžimas išdėstytas taip:

{"container": "<užduotis>", "cluster": "<algoritmas>"},

kur <užduotis> yra „Docker“ konteinerio, kuriame yra užduoties vykdymo failai, pavadinimas ir parametrai (sprendimas pagrįstas [90]). <algoritmas>

gali būti užduoties paskirstymo algoritmo arba klasterio, kuriam paskiriama vykdyti užduotį, pavadinimas. Pavyzdžiai:

- {"container": "ashael/pi 100000", "cluster": "FIFO"};
- {"container": "ashael/pi 200000", "cluster": "TSB-static(k=10)"};
- {"container": "-e \"INPUT_FILENAMES=1.json; 2.json\" -v /var/data/:/data -v /var/ out/:/out mrquad/map-reduce", "cluster": "mesos"}.

„Docker“ konteineriai yra saugomi viešosiose arba privačiosiose saugyklose. Šis sprendimas leidžia platformai veikti heterogeniškoje aplinkoje. Be to, jis sumažina tinklu perduodamų duomenų srautą, nes skaičiavimus atliekantys kompiuteriai „Docker“ konteinerius atsisiunčia tik vieną kartą, o ne kiekvieną kartą gavę užduotį iš planuoklio.

3.6 Skyriaus išvados

Šiame skyriuje pristatyti sukurti algoritmai, skirti apskaičiuoti užduočių sulaikymo buferio ilgį hibridinių paskirstytųjų skaičiavimų platformoje, naudojančioje BOINC ir „Apache Mesos“ klasterius. Kadangi pavyko sėkmingai realizuoti visus reikiamus algoritmus, tai leidžia daryti šias išvadas:

1. paskirstytųjų skaičiavimų užduotis įmanoma atlikti naudojant vidinius įmonės serverius ir asmeninius darbuotojų kompiuterius, be jokios papildomos informacijos apie užduotis poreikio;
2. įmanoma pritaikyti užduočių sulaikymo buferį hibridinių paskirstytųjų skaičiavimų platformose.

4. KOMPIUTERINIAI EKSPERIMENTAI IR KOMPIUTERINIS MODELIAVIMAS

Šiame skyriuje tiriama sukurta hibridinių paskirstytųjų skaičiavimų platforma, atliekanti paskirstytųjų skaičiavimų užduotis ir naudojanti privačiojo bei viešojo skaičiavimo išteklius (įmonės serverius ir asmeninius darbuotojų kompiuterius). Sukurta platforma naudoja užduočių paskirstymo algoritmą, turintį užduočių sulaikymo buferį. Tad taip pat tiriamas šio užduočių paskirstymo algoritmo efektyvumas ir taikymo daugiakanalėse sistemose galimybės. Šiame skyriuje siekiama įrodyti, kad siūlomas užduočių paskirstymo algoritmas pagerina platformos efektyvumą, lyginant su standartiniu FIFO algoritmu. Remiantis 3 skyriuje atlikta užduočių paskirstymo algoritmų apžvalga, FIFO yra vienintelis plačiai naudojamas algoritmas, galintis paskirstyti užduotis dviem ir daugiau skaičiavimo tinklų be jokios papildomos informacijos apie vykdomas užduotis poreikio.

4.1 Dviejų kanalų sistemos kompiuterinis modeliavimas

Dviejų kanalų sistemos kompiuterinio modeliavimo tyrimo tikslas – patikrinti keliamą hipotezę, kad siūlomas hibridinių paskirstytųjų skaičiavimų platformoje naudojamas užduočių paskirstymo algoritmas su užduočių sulaikymo buferiu pagerina užduočių vykdymo trukmę, lyginant su standartiniu FIFO algoritmu.

Tyrimams pasitelkiama virtualioji aplinka, sukurta naudojant PHP programavimo kalbą. Užduočių vykdymo trukmė matuojama naudojant visoms užduotims atlikti reikalingą iteracijų skaičių. Skirtingai nuo realios platformos eksperimentų (pateiktų 4.3 skyriuje), virtualioji aplinka imituoja infrastruktūrą, turinčią didesnį skaičiavimo mazgų kiekį, ir gali atlikti didelius tyrimų kiekius per priimtiną laiką. Šioje aplinkoje imituojama privačiųjų ir viešųjų skaičiavimo išteklių elgsena, tačiau nėra atsižvelgiama į duomenų perdavimo laiką ir tinklo apkrovos pokyčius.

Šis dviejų kanalų sistemos kompiuterinio modeliavimo tyrimas lygina užduočių aibės įvykdymo trukmę taikant šiuos skirtingus užduočių paskirstymo algoritmus:

- FIFO;
- TSB-static: hibridinių paskirstytųjų skaičiavimų platformos užduočių paskirstymo algoritmas, naudojantis statinio ilgio užduočių sulaikymo buferį. Buferio ilgis įvertinamas tik vieną kartą kiekvienam skaičiavimo tinklui atlikus bent po vieną užduotį.

- TSB-dynamic: hibridinių paskirstytųjų skaičiavimų platformos užduočių paskirstymo algoritmas, naudojantis dinaminio ilgio užduočių sulaikymo buferį. Buferio ilgis yra iš naujo įvertinamas sistemai gavus naują užduotį.

4.1.1 Kompiuterinio modeliavimo scenarijai

Siekiant tiksliai įvertinti užduočių vykdymo trukmę, sukurti įvairūs galimi scenarijai, kuriuose užduotys aprašomos naudojant šiuos du parametrus: užduočiai atlikti reikiamų ciklo iteracijų (pseudo operacijų) kiekį ir užduoties patekimo į sistemą trukmę. Trukmė taip pat matuojama iteracijomis. Visi scenarijai sugeneruoti prieš vykdant eksperimentus, kad visi užduočių paskirstymo algoritmai būtų ištirti vienodomis sąlygomis. Scenarijams aprašyti naudojamos šios anotacijos:

- $\boldsymbol{TS}$: statinio dydžio užduotys. Visos užduotys yra vienodo dydžio (200 iteracijų).
- $\boldsymbol{TD}$: dinaminio dydžio užduotys. Sugeneruoti užduočių dydžiai paskirstomi naudojant Puasono skirstinį ($\boldsymbol{\lambda = 200}$).
- $\boldsymbol{STS}$: statinio intensyvumo užduočių srautas. Kiekvienos naujos užduoties patekimo į sistemą laiką nuo pirmesnės skiria 8 iteracijos.
- $\boldsymbol{DTS}$: dinaminio intensyvumo užduočių srautas. Kiekvienos naujos užduoties patekimo į sistemą laiką nuo pirmesnės skiria kintantis iteracijų kiekis. Galimos reikšmės sugeneruotos naudojant Puasono skirstinį ($\boldsymbol{\lambda = 8}$).

Kiekvienam užduočių paskirstymo algoritmui ištirti naudojami šie scenarijai:

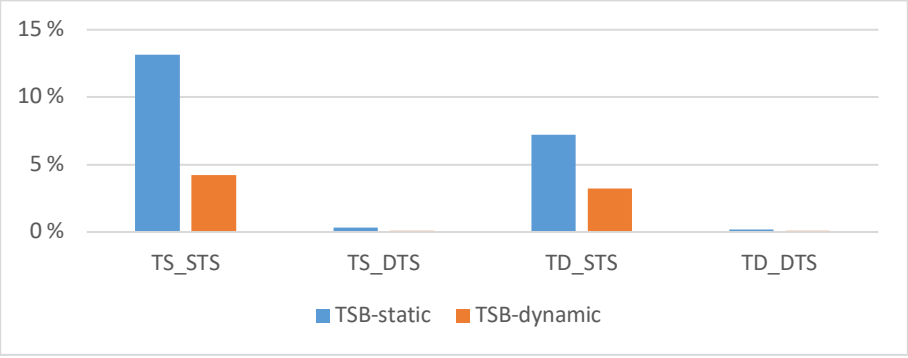
- $\boldsymbol{TS\_STS}$: statinio dydžio užduotys ($\boldsymbol{TS}$), statinio intensyvumo užduočių srautas ($\boldsymbol{STS}$). Vienodos užduotys platformai pateikiamos vienodais intervalais.
- $\boldsymbol{TS\_DTS}$: statinio dydžio užduotys ($\boldsymbol{TS}$), dinaminio intensyvumo užduočių srautas ($\boldsymbol{DTS}$). Vienodos užduotys platformai pateikiamos kintančiais intervalais.
- $\boldsymbol{TD\_STS}$: dinaminio dydžio užduotys ($\boldsymbol{TD}$), statinio intensyvumo užduočių srautas ($\boldsymbol{STS}$). Kintančios užduotys platformai pateikiamos vienodais intervalais.
- $\boldsymbol{TD\_DTS}$: dinaminio dydžio užduotys ($\boldsymbol{TD}$), dinaminio intensyvumo užduočių srautas ($\boldsymbol{DTS}$). Kintančios užduotys platformai pateikiamos kintančiu intervalu.

Iteracijų skaičius ir intervalai tarp naujų užduočių pritaikyti modeliuojamų skaičiavimo mazgų skaičiui. Norint pasiekti geriausių rezultatų, užduočių sulaikymo buferis neturėtų būti nuolat tuščias ar perpildytas. Priešingu atveju visos užduotys būtų nukreipiamos į privatųjį tinklą (sistema būtų nepakankamai išnaudojama) arba užduočių paskirstymo algoritmas elgtųsi lygiai taip pat, kaip ir standartinis FIFO algoritmas.

Kiekvienas scenarijus vykdomas naudojant visus įmanomus užduočių skaičius, pradedant nuo 40 iki 400 užduočių. Kiekvieno scenarijaus metu gauti rezultatai agreguojami. Atliekamuose tyrimuose lėtąjį kanalą aptarnauja 16 agentų, o greitąjį kanalą – 8 agentai. Prieš vykdydami naują užduotį, lėtojo kanalo agentai laukia 1000 iteracijų (simuliuojamas virtualiosios mašinos paleidimo laikas). Jie nustatyti vykdyti užduotis 10 kartų lėčiau už greitąjį kanalą aptarnaujančius agentus.

4.1.2 Kompiuterinio modeliavimo rezultatai

Apibendrinti kompiuterinio modeliavimo rezultatai pateikti 34 pav. ir 4 lentelėje. Rezultatus sudaro 4320 eksperimentų, kuriuose tiriami 3 algoritmai naudojant 4 skirtingus scenarijus su įvairiais užduočių kiekiais (nuo 40 iki 400). Rezultatai rodo, kad TSB algoritmas yra efektyviausias, esant vienodam užduočių srauto intensyvumui. Geriausias rezultatas pasiekiamas esant TS_STS scenarijui – gaunamas iki 13 % pagerėjimas, lyginant su standartiniu FIFO algoritmu. Gauti rezultatai leidžia daryti išvadą, kad užduočių sulaikymo buferis gali būti taikomas hibridinių paskirstytųjų skaičiavimų platformose, nes visais atvejais veikia ne blogiau už standartinį FIFO algoritmą (žr. 34 pav.).

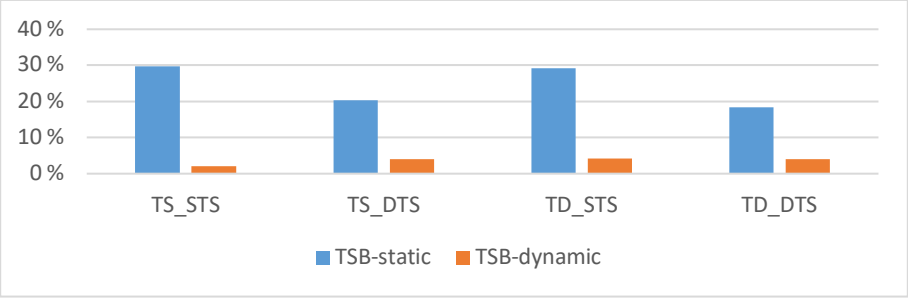


34 pav. Užduočių vykdymo laiko trumpėjimas lyginant su FIFO

4 lentelė. Kompiuterinio modeliavimo rezultatai

Algoritmas	Scenarijus	Vykdymo laikas (iteracijos)	Vykdymo laiko pagerėjimas lyginant su FIFO
FIFO	TS_STS	6138	-
	TS_DTS	236231	-
	TD_STS	5661	-
	TD_DTS	235794	-
TSB-static	TS_STS	5330	13,16 %
	TS_DTS	235450	0,33 %
	TD_STS	5252	7,22 %
	TD_DTS	235371	0,18 %
TSB-dynamic	TS_STS	5879	4,22 %
	TS_DTS	235997	0,1 %
	TD_STS	5478	3,23 %
	TD_DTS	235583	0,09 %

Tyrimo duomenų analizė parodė, kad TSB-static algoritmas į lėtąjį kanalą nukreipia 24,43 % mažiau užduočių už FIFO (žr. 35 pav.). TSB-dynamic algoritmas atitinkamai nukreipia 3,61 % mažiau užduočių už FIFO. Tai reiškia, kad TSB-static algoritmas užduočių įstrigimo problemą sprendžia geriau už TSB-dynamic algoritmą. TSB-dynamic algoritmas nukreipia vidutiniškai 21,57 % mažiau užduočių už TSB-static algoritmą.



35 pav. Į lėtąjį kanalą nukreipiamų užduočių kiekio mažėjimas lyginant su FIFO algoritmu

4.2 Daugiakanalės sistemos kompiuterinio modeliavimo tyrimas

Daugiakanalės sistemos kompiuterinio modeliavimo tyrimo tikslas – patikrinti keliamą hipotezę, kad siūlomas hibridinių paskirstytųjų skaičiavimų užduočių paskirstymo algoritmas su užduočių sulaikymo buferiu gerina užduočių vykdymo trukmę, lyginant su standartiniu FIFO algoritmu.

Tyrimams naudojama ta pati virtualioji aplinka ir scenarijai kaip ir 4.1 skyriuje. Šiam tyrimui naudojama virtualioji aplinka nustatyta 4 klasterių režimu. Papildomai atlikti tyrimai, siekiant patikrinti modeliuojamos sistemos efektyvumą esant skirtingam klasterių našumui.

5 lentelėje pateikti tyrimo rezultatai, kuriame naudojami skirtingo našumo klasteriai:

- A klasteris: 2 agentai;
- B klasteris: 2 agentai, 3 kartus lėtesni už A klasterio;
- C klasteris: 2 agentai, 9 kartus lėtesni už A klasterio;
- D klasteris: 2 agentai, 27 kartus lėtesni už A klasterio, kiekvieną naują užduotį pradeda vykdyti po 1000 iteracijų delsos (simuliuojamas virtualiosios mašinos paleidimo laikas).

Kiekvienas scenarijus vykdomas naudojant visus įmanomus užduočių skaičius, pradedant nuo 10 iki 100 užduočių. Kiekvieno scenarijaus metu gauti rezultatai agreguojami ir apskaičiuojamas gautų rezultatų vidurkis. Tyrimo rezultatai publikuoti [59].

5 lentelė. Tyrimo rezultatai. Naudojami skirtingo našumo klasteriai

Algoritmas	Scenarijus	Vykdymo laikas (iteracijos)	Vykdymo laiko pagerėjimas lyginant su FIFO
FIFO	TS_STS	7483	-
	TS_DTS	21702	-
	TD_STS	7940	-
	TD_DTS	22140	-
TSB-static	TS_STS	6919	7,54 %
	TS_DTS	21139	2,59 %
	TD_STS	7157	9,86 %
	TD_DTS	21387	3,4 %
TSB-dynamic	TS_STS	6919	7,54 %
	TS_DTS	21139	2,59 %
	TD_STS	7075	10,89 %
	TD_DTS	21320	3,7 %

6 lentelėje pateikti tyrimo, kuriame naudojami 2 skirtingo ir 2 vienodo našumo klasteriai, rezultatai:

- A klasteris: 2 agentai;
- B klasteris: 2 agentai, 6 kartus lėtesni už A klasterio;

- C klasteris: 2 agentai, 6 kartus lėtesni už A klasterio;
- D klasteris: 2 agentai, 27 kartus lėtesni už A klasterio, kiekvieną naują užduotį pradeda vykdyti po 1000 iteracijų delsos (simuliuojamas virtualiosios mašinos paleidimo laikas).

Kiekvienas scenarijus vykdomas naudojant visus įmanomus užduočių skaičius, pradedant nuo 10 iki 100 užduočių. Kiekvieno scenarijaus metu gauti rezultatai yra agreguojami.

6 lentelė. Tyrimo rezultatai. Naudojami 2 skirtingo ir 2 vienodo našumo klasteriai

Algoritmas	Scenarijus	Vykdymo laikas (iteracijos)	Vykdymo laiko pagerėjimas lyginant su FIFO
FIFO	TS_STS	7576	-
	TS_DTS	21795	-
	TD_STS	7809	-
	TD_DTS	22025	-
TSB-static	TS_STS	6840	9,71 %
	TS_DTS	21058	3,38 %
	TD_STS	7307	6,43 %
	TD_DTS	21484	2,46 %
TSB-dynamic	TS_STS	6840	9,71 %
	TS_DTS	21058	3,38 %
	TD_STS	7062	9,57 %
	TD_DTS	21280	3,38 %

7 lentelėje pateikti tyrimo, kuriame naudojami skirtingo našumo klasteriai, rezultatai:

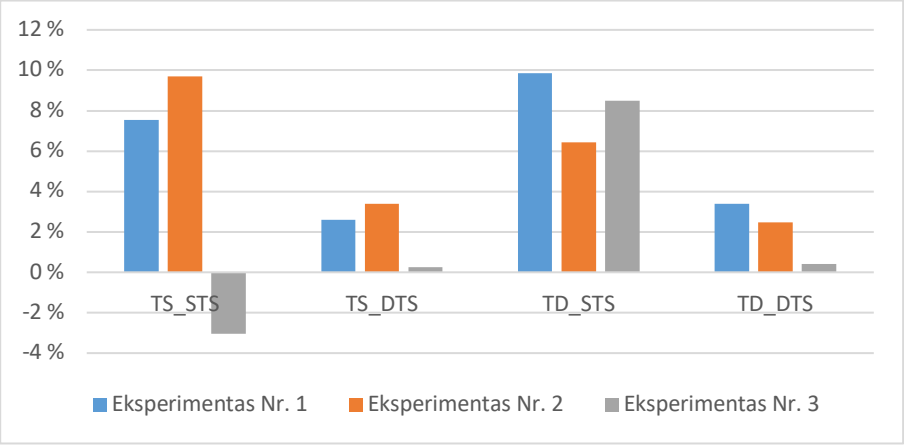
- A klasteris: 5 agentai;
- B klasteris: 5 agentai, 3 kartus lėtesni už A klasterio;
- C klasteris: 5 agentai, 6 kartus lėtesni už A klasterio;
- D klasteris: 20 agentų. Agentai, 27 kartus lėtesni už A klasterio, kiekvieną naują užduotį pradeda vykdyti po 1000 iteracijų delsos (simuliuojamas virtualiosios mašinos paleidimo laikas).

Kiekvienas scenarijus vykdomas naudojant visus įmanomus užduočių skaičius, pradedant nuo 40 iki 400 užduočių. Kiekvieno scenarijaus metu gauti rezultatai yra agreguojami.

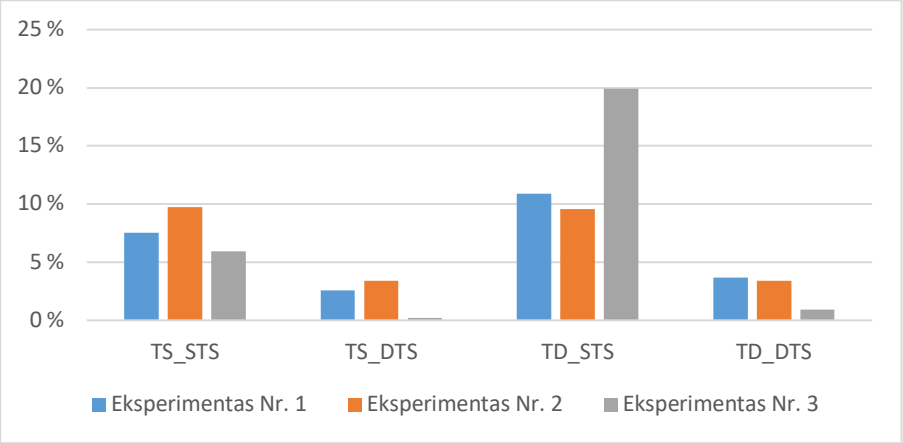
7 lentelė. Tyrimo rezultatai. Naudojami skirtingo našumo klasteriai

Algoritmas	Scenarijus	Vykdymo laikas (iteracijos)	Vykdymo laiko pagerėjimas lyginant su FIFO
FIFO	TS_STS	9520	-
	TS_DTS	239643	-
	TD_STS	10630	-
	TD_DTS	240739	-
TSB-static	TS_STS	9810	-3,05 %
	TS_DTS	239029	0,26 %
	TD_STS	9727	8,49 %
	TD_DTS	239780	0,4 %
TSB-dynamic	TS_STS	8956	5,92 %
	TS_DTS	239102	0,23 %
	TD_STS	8513	19,92 %
	TD_DTS	238569	0,9 %

Pateiktų tyrimų rezultatai rodo, kad siūlomas užduočių paskirstymo algoritmas su dinaminio ilgio užduočių sulaikymo buferiu visais atvejais gerina užduočių vykdymo trukmę lyginant su standartiniu FIFO algoritmu (žr. 36 ir 37 pav.). Taikant TSB-static algoritmą TS_STS scenarijaus atveju gautas neigiamas rezultatas, nes buvo naudojamas per ilgas užduočių sulaikymo buferis. Tad lėtojo kanalo skaičiavimo pajėgumai buvo naudojami nepakankamai ir užduočių atlikimo laikas pailgėjo.



36 pav. Užduočių vykdymo laiko trumpėjimas naudojant užduočių paskirstymo metodą su nekintančio ilgio buferiu (lyginant su FIFO)



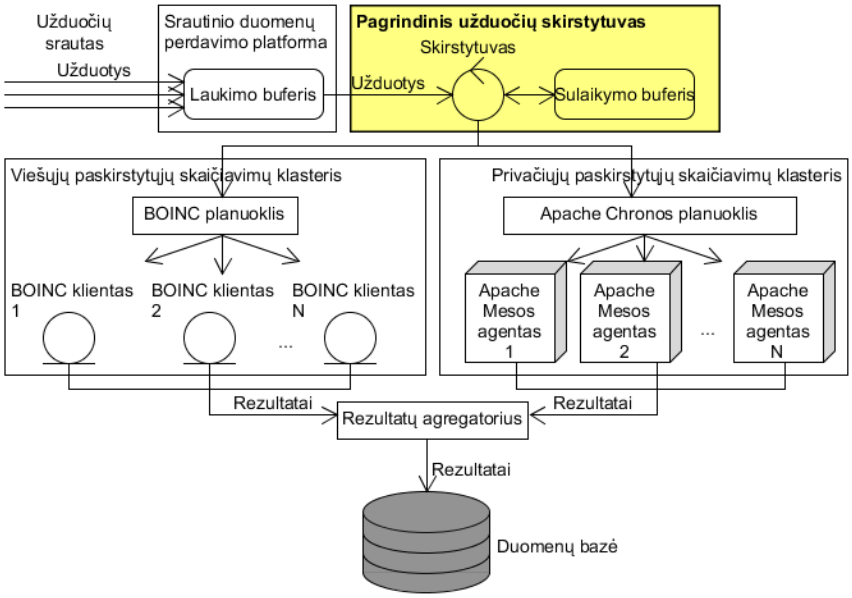
37 pav. Užduočių vykdymo laiko trumpėjimas naudojant užduočių paskirstymo metodą su kintančio ilgio buferiu (lyginant su FIFO)

Tyrimų rezultatai parodė, kad TSB-dynamic algoritmas užduočių eilės vykdymo laiką sumažina vidutiniškai 2,29 % daugiau už TSB-static algoritmą.

4.3 Platformos efektyvumo tyrimas

Šiame skyriuje tiriama sukurta hibridinių paskirstytųjų skaičiavimų platforma (pristatyta 3.5.1 skyriuje, žr. 38 pav.). Aprašomas eksperimentas, kuriuo siekiama patikrinti, ar sukurtos platformos tyrimų rezultatai sutampa su kompiuterinio modeliavimo tyrimų rezultatais. Siekiama ištirti ir palyginti standartinio užduočių paskirstymo algoritmo FIFO ir toliau nurodytų dviejų algoritmų užduočių vykdymo suminį laiką (angl. *makespan*):

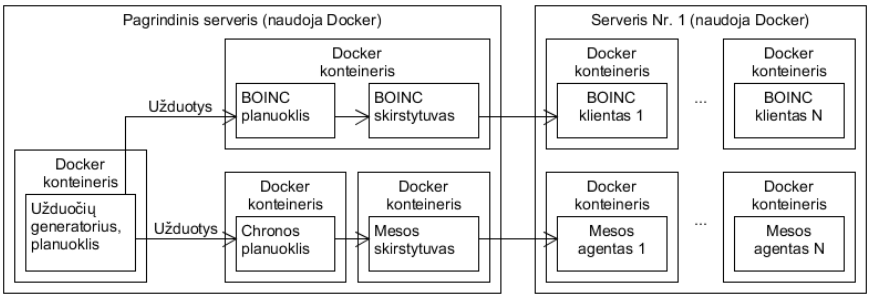
- TSB-static (k): hibridinių paskirstytųjų skaičiavimų platformos užduočių paskirstymo algoritmas, naudojantis statinio ilgio užduočių sulaikymo buferį, kur *k* yra buferio ilgis (buferio ilgis įvertinamas tik vieną kartą kiekvienam skaičiavimo tinklui atlikus bent po vieną užduotį).
- TSB-dynamic: hibridinių paskirstytųjų skaičiavimų platformos užduočių paskirstymo algoritmas, naudojantis dinaminio ilgio užduočių sulaikymo buferį (buferio ilgis yra iš naujo įvertinamas sistemai gavus naują užduotį).



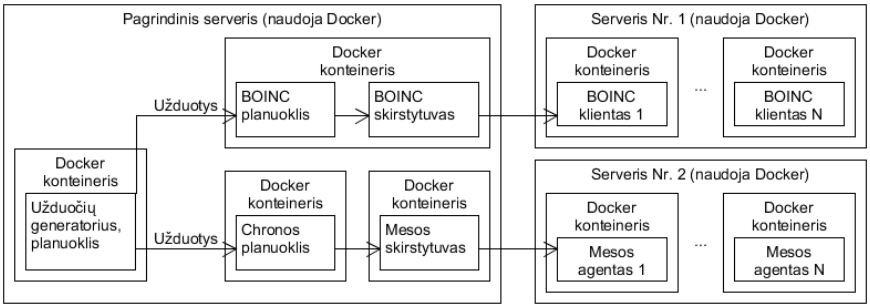
38 pav. Architektūra, sujungianti du klasterius naudojant dviejų lygių planuoklių hierarchiją

4.3.1 Serverių sąrankos

Tyrimuose naudojamos dvi skirtingos tyrimams skirtų serverių sąrankos (A ir B serverių sąrankos). Naudojamos dvi aplinkos siekiant įsitikinti, kad skirtingose aplinkose tyrimų rezultatai bus panašūs. A sąranka sudaryta iš dviejų atskirų serverių, kuriuose veikia „Docker“ konteineriai (žr. 39 pav.). Pagrindinis serveris (angl. *master*) skirtas klasteriams valdyti ir užduotims paskirstyti. Antrinis (angl. *slave*) serveris skirtas dviejų klasterių emuliavimui naudojant virtualiąsias mašinas, aptarnaujančias atskirus skaičiavimo mazgus. Kadangi abu klasteriai yra viename serveryje, atliekama papildoma virtualiųjų mašinų konfigūracija, nustatanti viršutines atminties (RAM) ir procesoriaus išteklių naudojimo ribas. Riboms nustatyti keičiami „Docker“ konteinerių, „VirtualBox“ atvaizdų (reikalingų BOINC klientams vykdyti), „Apache Mesos“ agentų ir BOINC klientų parametrai. Šie ribojimai kontroliuoja išteklių naudojimą ir užtikrina vienodą išteklių paskirstymą kiekvienai užduočiai.



39 pav. A serverių sąrankos schema



40 pav. B serverių sąrankos schema

B serverių sąranka yra labai panaši į A serverių sąranką. Šios sąrankos skiriasi tuo, kad B sąrankoje vietoje vieno antrinio serverio naudojami du antriniai serveriai (žr. 40 pav.). Taip atskiriami du klasteriai ir gaunami papildomi ištekliai, panaudojami emuliuojant daugiau skaičiavimuose dalyvaujančių kompiuterių. Abiejose sąrankose emuliuojamų mazgų skaičius apribojamas virtualiąsias mašinas apimančio serverio branduolių skaičiumi. Naudojant A serverių sąranką, emuliuojami du „Apache Mesos“ agentai ir du BOINC klientai. Naudojant B serverių sąranką, emuliuojami du „Apache Mesos“ agentai ir keturi BOINC klientai. Naudojant dvi serverių sąrankas galima patikrinti, ar pridedant daugiau skaičiavimuose dalyvaujančių kompiuterių į viešojo skaičiavimo tinklą keičiasi rezultatai. Abiejose sąrankose naudojami užduočių generatoriai užduočių srautui emuliuoti.

Tyrimams naudojamų serverių specifikacija:

- pagrindinis serveris: Intel(R) Xeon(R) 5160 CPU, 2 branduoliai, 3,00 GHz, 8 GB darbinė atmintis, 500 GB kietasis diskas, Ubuntu Linux 16.04.6 operacinė sistema su 4.15.0-88-generic branduoliu (x86_64);
- antrinis serveris Nr. 1: Intel(R) Core(TM) i5-4460 CPU, 4 branduoliai, 3.20 GHz, 24 GB darbinė atmintis, 450 GB kietasis

diskas, Ubuntu Linux 18.04.4 operacinė sistema su 4.15.0-91-generic branduoliu (x86_64);

- antrinis serveris Nr. 2: Intel(R) Core(TM) i7-6700HQ CPU, 4 branduoliai, 2.60 GHz, 16 GB darbinė atmintis, 128 GB kietasis diskas, Linux Fedora 31 operacinė sistema su 5.3.16-300.fc31.x86_64 branduoliu (x86_64).

4.3.2 Eksperimentų scenarijai

Siekiant įvertinti užduočių vykdymo suminį laiką (angl. *makespan*), įvykdytos paskirstytųjų skaičiavimų užduotys, vertinančios π reikšmę „Monte Karlo"[28] metodu. Ši užduotis reikalauja tik procesoriaus pajėgumų, todėl leidžia tiksliau įvertinti užduoties vykdymo laiką lyginant skirtingus užduočių paskirstymo algoritmus. Tokiu būdu pašalinami kiti šalutiniai veiksniai, įskaitant tinklo pralaidumą ir duomenų saugojimo spartą. Svarbu paminėti, kad sukurta hibridinių paskirstytųjų skaičiavimų platforma gali būti naudojama įvairaus pobūdžio paskirstytųjų skaičiavimų užduotims vykdyti.

Scenarijams aprašyti naudojamos šios anotacijos:

- $\boldsymbol{TS(i)}$ – statinio dydžio užduotys, kur $\boldsymbol{i}$ yra vienai užduočiai skirtų iteracijų skaičius π reikšmei apskaičiuoti;
- $\boldsymbol{TD(I)}$ – dinaminio dydžio užduotys, kur $\boldsymbol{I}$ yra galimų užduočių dydžių rinkinys. Galimų dydžių parinkimo tikimybės paskirstomos naudojant diskretųjį vienodą paskirstymą (angl. *discrete uniform distribution*) ($\boldsymbol{a = 0}$, $\boldsymbol{b = 1}$);
- $\boldsymbol{STS(d)}$ – statinis užduočių srauto intensyvumas, kur $\boldsymbol{d}$ – intervalas sekundėmis tarp į sistemą patenkančių naujų užduočių;
- $\boldsymbol{DTS(D)}$ – dinaminis užduočių srauto intensyvumas, kur $\boldsymbol{D}$ – gaunamų užduočių galimų intervalų rinkinys. Galimų intervalų parinkimo tikimybės paskirstomos naudojant Puasono skirstinį ($\boldsymbol{\lambda = 30}$).

Kiekvienam užduočių paskirstymo algoritmui ištirti naudojami šie scenarijai:

- $\boldsymbol{TS\_STS}$: statinio dydžio užduotys $\boldsymbol{TS(100000)}$, statinis užduočių srauto intensyvumas $\boldsymbol{STS(60)}$;
- $\boldsymbol{TS\_DTS}$: statinio dydžio užduotys $\boldsymbol{TS(100000)}$, dinaminis užduočių srauto intensyvumas $\boldsymbol{DTS(\{60, 120\})}$;

[28] https://hub.docker.com/r/ashael/pi

- $\boldsymbol{TD\_STS}$: dinaminio dydžio užduotys $\boldsymbol{TD}(\{\mathbf{100000}, \mathbf{200000}\})$, statinis užduočių srauto intensyvumas $\boldsymbol{STS}(\mathbf{60})$;
- $\boldsymbol{TD\_DTS}$: dinaminio dydžio užduotys $\boldsymbol{TD}(\{\mathbf{100000}, \mathbf{200000}\})$, dinaminis užduočių srauto intensyvumas $\boldsymbol{DTS}(\{\mathbf{60}, \mathbf{120}\})$.

Iteracijų skaičius π reikšmei apskaičiuoti ir intervalų ilgio tarp į sistemą patenkančių užduočių parametrai pritaikyti eksperimentuose naudojamai aparatinei įrangai. Siekiant gauti geriausių rezultatų, užduočių sulaikymo buferis neturėtų būti nuolat tuščias arba perpildytas. Priešingu atveju visos užduotys yra nukreiptos į privačiųjų skaičiavimų klasterį (sistema nepakankamai išnaudojama) arba užduočių paskirstymo algoritmas veikia lygiai taip pat, kaip ir standartinis FIFO algoritmas.

Eksperimentų scenarijai aprašomi naudojant sąrankų bylas. 41 pav. pateiktas vienos iš tokių bylų pavyzdys. Kiekviena scenarijaus sąrankos bylos eilutė atitinka atskirą eksperimentą. Eilutę sudaro du eksperimento parametrai, atskirti kabliataškiu (žr. 41 pav.):

1. eksperimento pavadinimas, naudojamas eksperimento rezultatų rinkimui automatizuoti;
2. eksperimento paleidimo komanda, naudojama nustatyti eksperimento parametrams.

100_tasks/experiment.csv

```
vs01;./bin/scheduler -g -m 60000 -n 100 -t '{"container": "ashael/pi
100000", "cluster": "first-available"}'
vs02;./bin/scheduler -g -b 10 -m 60000 -n 100 -t '{"container": "ashael/pi
100000", "cluster": "stalling-buffer"}'
vs03;./bin/scheduler -g -m 60000 -n 100 -t '{"container": "ashael/pi
100000", "cluster": "stalling-buffer"}'
vs04;./bin/scheduler -g -w from-file -f ~/100_tasks/first-available-
static60_dynamic-tasks.txt
vs05;./bin/scheduler -g -b 10 -w from-file -f ~/100_tasks/stalling-
static60_dynamic-tasks.txt
vs06;./bin/scheduler -g -w from-file -f ~/100_tasks/stalling-
static60_dynamic-tasks.txt
vs07;./bin/scheduler -g -w from-file -f ~/100_tasks/first-available-
30_static-tasks.txt
vs08;./bin/scheduler -g -b 10 -w from-file -f ~/100_tasks/stalling-
30_static-tasks.txt
vs09;./bin/scheduler -g -w from-file -f ~/100_tasks/stalling-30_static-
tasks.txt
```

```
vs10;./bin/scheduler -g -w from-file -f ~/100_tasks/first-available-30.txt
vs11;./bin/scheduler -g -b 10 -w from-file -f ~/100_tasks/stalling-30.txt
vs12;./bin/scheduler -g -w from-file -f ~/100_tasks/stalling-30.txt
```

41 pav. Scenarijaus sąrankos bylos pavyzdys

Eksperimento paleidimo komanda naudoja sukurtos hibridinių paskirstytųjų skaičiavimų platformos užduočių paskirstymo modulį, kuris per eksperimentus nustatomas užduočių generavimo režimu (naudojant $-g$ parametrą). Skirtingi scenarijai sudaromi naudojant šiuos parametrus (žr. 42 pav.):

1. $-\boldsymbol{b}$ – statinio užduočių sulaikymo buferio ilgis;
2. $-\boldsymbol{m}$ – intervalas (milisekundėmis) tarp sugeneruotų užduočių patekimo į sistemą;
3. $-\boldsymbol{n}$ – užduočių kiekis;
4. $-\boldsymbol{t}$ – užduoties apibrėžimas;
5. $-\boldsymbol{w}$ $-\boldsymbol{f}$ – šie parametrai nurodo, kad eksperimente naudojami kintantys sugeneruotų užduočių patekimo į sistemą intervalai ir skirtingi užduočių apibrėžimai, tad turi būti naudojama papildoma sąrankos byla (žr. 42 pav.), kurioje pateikiamas intervalų ir užduočių apibrėžimų (atskirtų kabliataškiu) sąrašas.

100_tasks/stalling-30.txt

```
{"delay": 120, "task": "{\"container\": \"ashael/pi 100000\", \"cluster\": \"stalling-buffer\"}"}
{"delay": 60, "task": "{\"container\": \"ashael/pi 100000\", \"cluster\": \"stalling-buffer\"}"}
{"delay": 120, "task": "{\"container\": \"ashael/pi 100000\", \"cluster\": \"stalling-buffer\"}"}
{"delay": 60, "task": "{\"container\": \"ashael/pi 200000\", \"cluster\": \"stalling-buffer\"}"}
{"delay": 60, "task": "{\"container\": \"ashael/pi 100000\", \"cluster\": \"stalling-buffer\"}"}
{"delay": 60, "task": "{\"container\": \"ashael/pi 100000\", \"cluster\": \"stalling-buffer\"}"}
{"delay": 60, "task": "{\"container\": \"ashael/pi 100000\", \"cluster\": \"stalling-buffer\"}"}
{"delay": 60, "task": "{\"container\": \"ashael/pi 100000\", \"cluster\": \"stalling-buffer\"}"}
```

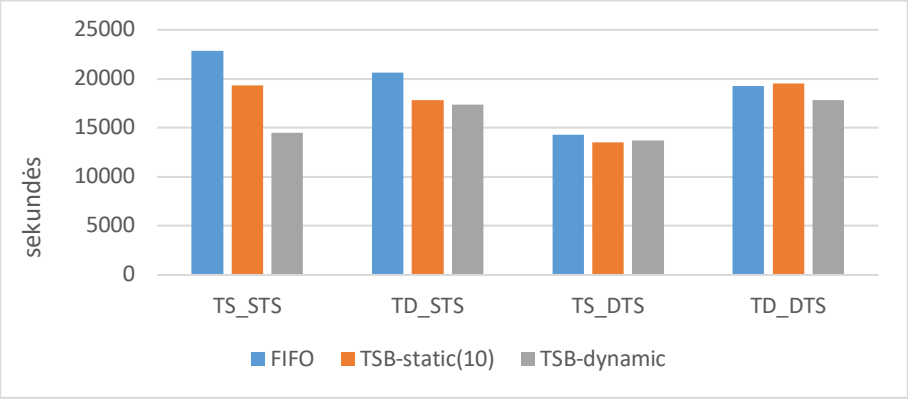
{"delay": 120, "task": "{\"container\": \"ashael/pi 100000\", \"cluster\": \"stalling-buffer\"}"}

42 pav. Intervalų ir užduočių apibrėžimų sąrašas

Po kiekvieno eksperimento sistema atstatoma į pradinę būseną, ištrinant eksperimente dalyvavusių virtualiųjų mašinų atvaizdus.

4.3.3 Tyrimų rezultatai naudojant A sąranką

Šio tyrimo tikslas – gauti pirminius hibridinių paskirstytųjų skaičiavimų platformoje įgyvendintų užduočių paskirstymo algoritmų efektyvumo rezultatus. Naudojant A sąranką atlikta 100 užduočių. Gauti rezultatai rodo (žr. 43 pav. ir 8 lentelę), kad užduočių sulaikymo buferis trumpina užduočių vykdymo laiką. Statinio ilgio užduočių sulaikymo buferio naudojimas gerina rezultatus vidutiniškai 8,875 %, lyginant su standartiniu FIFO algoritmu. O dinaminio ilgio užduočių sulaikymo buferis gerina rezultatus vidutiniškai 17,63 %.



43 pav. 100 π reikšmės vertinimo užduočių, sudarytų naudojant „Monte Karlo“ metodą, suminis vykdymo laikas (angl. „makespan“)

8 lentelė: Užduočių vykdymo laiko trumpėjimas lyginant su FIFO

Algoritmas	Scenarijus	Vykdymo laikas (sekundės)	Vykdymo laiko trumpėjimas lyginant su FIFO
FIFO	TS_STS	22863	-
	TD_STS	20627	-
	TS_DTS	14318	-
	TD_DTS	19251	-

TSB-static(10)	TS_STS	19331	15,45 %
	TD_STS	17832	13,55 %
	TS_DTS	13495	5,75 %
	TD_DTS	19562	-1,62 %
TSB-dynamic	TS_STS	14522	36,48 %
	TD_STS	17394	15,67 %
	TS_DTS	13727	4,13 %
	TD_DTS	17830	7,38 %

4.3.4 Tyrimų rezultatai naudojant B sąranką

Šio tyrimo tikslas – parodyti, kad naudojant B sąranką gaunami panašūs tyrimų rezultatai kaip ir A sąrankos atveju. Tyrime emuliuojamo viešųjų paskirstytųjų skaičiavimų klasteryje dalyvauja didesnis kiekis kompiuterių (4 virtualiosios mašinos). Siekiant gauti patikimų rezultatų, tyrimo metu atlikta daug daugiau eksperimentų, kiekvienam scenarijui naudojant skirtingus užduočių kiekius: 20, 40, 60. Be to, kiekvienas eksperimentas atliktas 5 kartus, siekiant įvertinti vidutinius laiko nuokrypius (angl. *average time deviations*) ir p-reikšmę.

Šis tyrimas naudoja nulinę hipotezę (dvipusę, α=0,05). Tokiu būdu siekiama įrodyti, kad yra bent 95 % tikimybė, jog alternatyvi hipotezė yra teisinga. Šiuo atveju nulinė hipotezė teigia, kad užduočių sulaikymo buferis neturi įtakos užduočių vykdymo trukmei. Taip tiriama, ar vidutinis visų užduočių įvykdymo laikas naudojant FIFO algoritmą skiriasi nuo vidutinio užduočių įvykdymo laiko naudojant siūlomus užduočių sulaikymo algoritmus:

- TSB-static (10);
- TSB-dynamic.

Agreguoti 180 eksperimentų rezultatai pateikti 9, 10 ir 11 lentelėse. Scenarijai, kuriuose siūlomi algoritmai veikė geriau už FIFO (reikšmingumas α = 0,05), paryškinti 10 ir 11 lentelėse.

9 lentelė: Vidutinis užduočių eilės įvykdymo laikas naudojant FIFO algoritmą

Scenarijus	Užduočių kiekis	Vidutinis vykdymo laikas (sek.)	Standartinis nuokrypis (sek.)
TS_STS	20	3598,4	189,2
	40	7063,6	212,86
	60	9995,4	223,01

TD_STS	20	5841,8	654,4
	40	9622,2	1229,63
	60	14905,4	1382,52
TS_DTS	20	3474	104,82
	40	6576	145,25
	60	9754	340,32
TD_DTS	20	5933,4	455,89
	40	10038,4	1439,31
	60	14620,4	1255,28

10 lentelė: Vidutinis užduočių eilės įvykdymo laikas naudojant TSB-static(10) algoritmą

Scenarijus	Užduočių kiekis	Vidutinis vykdymo laikas (sek.)	Standartinis nuokrypis (sek.)	Gerėjimas lyginant su FIFO	p-reikšmė lyginant su FIFO
TS_STS	20	3503,8	394,22	2,63 %	0,646
	40	**6377,8**	**32,8**	**9,71 %**	**0,002**
	60	**8978,6**	**297,52**	**10,17 %**	**0,0005**
TD_STS	**20**	**3967,4**	**959,76**	**32,09 %**	**0,009**
	40	9068,4	172,92	5,76 %	0,375
	60	13381,8	393,63	10,22 %	0,064
TS_DTS	20	3241,6	203	6,69 %	0,063
	40	**6168,8**	**256,39**	**6,19 %**	**0,021**
	60	**9254**	**240,38**	**5,13 %**	**0,031**
TD_DTS	**20**	**3127**	**824,78**	**47,3 %**	**0,001**
	40	8690	301,94	13,43 %	0,11
	60	**12293**	**1566,73**	**15,92 %**	**0,032**

11 lentelė: Vidutinis užduočių eilės įvykdymo laikas naudojant TSB-dynamic algoritmą

Scenarijus	Užduočių kiekis	Vidutinis vykdymo laikas (sek.)	Standartinis nuokrypis (sek.)	Gerėjimas lyginant su FIFO	p-reikšmė lyginant su FIFO
TS_STS	20	3493	133,24	2,93 %	0,342
	40	**6191,8**	**218,76**	**12,34 %**	**0,0002**
	60	**9147,8**	**264,42**	**8,48 %**	**0,001**
TD_STS	20	5016,4	117,66	14,13 %	0,05

	40	8759,8	257,83	8,96 %	0,2
	60	13256,2	480,22	11,06 %	0,053
TS_DTS	20	3497,4	133,06	-0,67 %	0,765
	40	**6342,2**	**88,81**	**3,56 %**	**0,018**
	60	**9204**	**118,35**	**5,64 %**	**0,019**
TD_DTS	**20**	**4697**	**607,02**	**20,84 %**	**0,008**
	40	9018,8	193,9	10,16 %	0,192
	60	13262,2	461,7	9,29 %	0,072

Rezultatai rodo 47,3 % trumpėjimą naudojant statinio ilgio užduočių sulaikymo buferį ir 20,84 % trumpėjimą naudojant dinaminio ilgio užduočių sulaikymo buferį, lyginant su standartiniu FIFO algoritmu. Statinio ilgio užduočių sulaikymo buferis pasiekia geresnių rezultatų vykdydamas 20 užduočių scenarijų, nes nė karto neperpildomas užduočių sulaikymo buferis. Neįmanoma efektyviai naudoti viešųjų paskirstytųjų skaičiavimų klasterio atliekant nedidelį skaičių užduočių, nes privačiųjų paskirstytųjų skaičiavimų klasterio pajėgumas viršija viešųjų paskirstytųjų skaičiavimų klasterio pajėgumus. Viešųjų paskirstytųjų skaičiavimų klasteris negavo jokių užduočių, tad platforma nebuvo gana apkrauta. Kadangi sistema nebuvo pakankamai išnaudota, į savo išvadas neįtrauksime šio eksperimento scenarijaus rezultatų.

Eksperimentai, atlikti apdorojant 60 užduočių ir naudojant bet kurį iš dviejų užduočių sulaikymo buferių parodė, kad 100 % atvejų rezultatai yra geresni už FIFO. Eksperimentuose su 40 užduočių statistiškai reikšmingas pagerėjimas gautas:

- 80 % atvejų naudojant TSB-static (10);
- 85 % atvejų naudojant TSB-dynamic.

Taip pat atliktas plataus masto eksperimentas, vykdant 200 užduočių. Rezultatai rodo iki 5,86 % pagerėjimą naudojant TSB-static (10) ir iki 6,31 % pagerėjimą naudojant TSB-dynamic užduočių paskirstymo algoritmą (žr. 12 lentelę).

12 lentelė: Užduočių vykdymo laiko trumpėjimas lyginant su FIFO

Algoritmas	Scenarijus	Vykdymo laikas (sek.)	Vykdymo laiko trumpėjimas lyginant su FIFO
FIFO	TS_STS	31361	-
	TD_STS	47502	-

	TS_DTS	32154	-
	TD_DTS	45159	-
TSB-static(10)	TS_STS	30504	2,73 %
	TD_STS	44718	5,86 %
	TS_DTS	30431	5,36 %
	TD_DTS	44226	2,07 %
TSB-dynamic	TS_STS	30700	2,11 %
	TD_STS	44504	6,31 %
	TS_DTS	30547	5 %
	TD_DTS	44963	0,43 %

4.4 Skyriaus išvados

Kompiuterinio modeliavimo ir empirinių tyrimų rezultatai rodo, kad užduočių sulaikymo buferis trumpina suminį visų užduočių vykdymo laiką. Kompiuterinio modeliavimo tyrimai parodė, kad siūlomas hibridinių paskirstytųjų skaičiavimų platformos užduočių paskirstymo algoritmas su nekintančiu užduočių sulaikymo buferiu užduočių suminį vykdymo laiką trumpina iki 47,3 %. Gauti tyrimų rezultatai leidžia daryti šias išvadas:

1. užduočių sulaikymo buferis gali būti taikomas hibridinių paskirstytųjų skaičiavimų sprendimams ir pagerinti darbo krūvio balansą tarp dviejų ir daugiau klasterių;
2. pritaikius užduočių sulaikymo buferį, užduočių eilė įvykdoma greičiau nei taikant FIFO algoritmą;
3. reikšmingiausias pagreitėjimas pasiekiamas vykdant mažą užduočių kiekį vidutiniškai apkrautoje sistemoje;
4. kai sistema yra labai apkrauta didesniu trumpų užduočių kiekiu, pastebėtas pagreitėjimas yra mažesnis;
5. TSB-dynamic algoritmas iš anksto neapibrėžtų užduočių eilę daugiakanalėje sistemoje įvykdo 2,29 % greičiau už TSB-static algoritmą;
6. TSB-static algoritmas į lėtąjį kanalą nukreipia vidutiniškai 24,43 % mažiau užduočių už FIFO algoritmą;
7. TSB-dynamic algoritmas nukreipia vidutiniškai 3,61 % mažiau užduočių už FIFO algoritmą;
8. TSB-dynamic algoritmas nukreipia vidutiniškai 21,57 % mažiau užduočių už TSB-static algoritmą.

IŠVADOS

Darbo išvados:

1. literatūros apžvalgos rezultatai parodė, kad viešųjų paskirstytųjų skaičiavimų platforma didelių duomenų apdorojimui, kurioje būtų išspręsta užduočių įstrigimo problema heterogeniškuose paskirstytųjų skaičiavimų tinkluose nenaudojant užduočių replikacijos arba informacijos apie užduočių dydžius, nėra sukurta;
2. remiantis kompiuterinio modeliavimo rezultatais, užduočių įstrigimo mažo našumo ištekliuose problema veiksmingai sprendžiama taikant sukurtą hibridinių paskirstytųjų skaičiavimų architektūrą, kai užduočių patekimo į sistemą srautas yra pastovus.
    a. TSB-static algoritmas į lėtąjį kanalą nukreipia vidutiniškai 24,43 % mažiau užduočių už FIFO algoritmą;
    b. TSB-dynamic algoritmas nukreipia vidutiniškai 3,61 % mažiau užduočių už FIFO algoritmą;
    c. TSB-dynamic algoritmas nukreipia vidutiniškai 21,57 % mažiau užduočių už TSB-static algoritmą.
3. daugiakanalės sistemos kompiuterinio modeliavimo tyrimas parodė, kad užduočių paskirstymo algoritmas, naudojantis dinaminio ilgio užduočių sulaikymo buferį, iš anksto neapibrėžtų užduočių eilę įvykdo 2,29 % greičiau už algoritmą, naudojantį statinio ilgio užduočių sulaikymo buferį;
4. sukurta hibridinių paskirstytųjų skaičiavimų platforma, naudojanti statinio ilgio užduočių sulaikymo buferį, iš anksto neapibrėžtų užduočių eilę įvykdo iki 47,3 % greičiau už FIFO algoritmą:
    a. reikšmingiausias pagreitėjimas pasiekiamas vykdant mažą užduočių kiekį vidutiniškai apkrautoje sistemoje;
    b. pastebėtas mažesnis pagreitėjimas, kai sistema yra labai apkrauta didesniu trumpų užduočių kiekiu.

LITERATŪROS SĄRAŠAS

1. Aazam, M., Hung, P., Huh, E. Cloud of Things: Integrating Internet of Things with Cloud Computing and the Issues Involved. 11th International Bhurban Conference on Applied Sciences and Technology, (IBCAST 2014), Islamabad, Pakistan, 414-419.
2. Abdi, S., PourKarimi, L., Ahmadi, M., Zargarid, F. Cost minimization for deadline-constrained bag-of-tasks applications in federated hybrid clouds. Future Generation Computer Systems, 2017, vol. 71, 113–128.
3. Alexandrov, A., Ibel, M., Schauser, K., Scheiman, C. Superweb: Towards a Global Web-Based Parallel Computing Infrastructure. In Parallel Processing Symposium, 1997, 100–106.
4. Ananthanarayanan, G., Ghodsi, A., Shenker, S., Stoica, I. Effective straggler mitigation: attack of the clones. Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation, (NSDI), 2013, 185–198.
5. Ananthanarayanan, G., Hung, M. C. C., Ren, X., Stoica, I., Wierman, A., Yu, M. GRASS: trimming stragglers in approximation analytics. 11th USENIX symposium on networked systems design and implementation, (NSDI 14), 2014, 289–302.
6. Ananthanarayanan, G., Kandula, S., Greenberg, A. G., Stoica, I., Yi, L., Saha, B., Harris, E. Reining in the outliers in map-reduce clusters using Mantri. 9th USENIX Symposium on Operating Systems Design and Implementation, (OSDI), Vancouver, BC, Canada, 2010, 10(1):24.
7. Anderson, D. Boinc Status Report. In: The 7th BOINC Workshop, 2011.
8. Anderson, D. BOINC: A System for Public-Resource Computing and Storage. Proceedings of Fifth IEEE/ACM International Workshop on Grid Computing, 2004, 410.
9. Anderson, D. P., Fedak, G. The Computational and Storage Potential of Volunteer Computing. Sixth IEEE International Symposium on Cluster Computing and the Grid, (CCGRID 06), 2006, 73–80.
10. Andrzejak, A., Domingues, P., Silva, L. Classifier-Based Capacity Prediction for Desktop Grids. CoreGRID Technical Report, (TR-0026), 2006.
11. Anjomshoa, M., Salleh, M. Overview on Clouds@home: Virtualization Mechanism for Volunteer Computing. Parallel and

Distributed Computing Systems, (PDCS 2014), Ukraine, Kharkiv, March 4–6, 2014, 11–19.
12. Anjomshoa, M., Salleh, M., Kermani., M. P. A Taxonomy and Survey of Distributed Computing Systems. Journal of Applied Sciences, 2015, 15(1), 46–57.
13. Baratloo, A., Karaul, M., Kedem, Z., Wijckoff, P. Charlotte: Metacomputing on the Web. Future Generation Computer Systems, 1999, 15(5–6), 559–570.
14. Barbalace, D., Lucchese, C., Mastroianni, C., Orlando, S., Talia, D. Mining@Home: Public Resource Computing for Distributed Data Mining, Concurrency & Computation: Practice & Experience, Wiley, 2010, 22, 658–682.
15. Bhatia, M. Task Scheduling in Grid Computing: A Review, Advances in Computational Sciences and Technology, ISSN 0973-6107, vol. 10(6), 2017, 1707–1714.
16. Bittencourt, L. F., Madeira, E. R. M., Fonseca, N. Scheduling in Hybrid Clouds. IEEE Communications Magazine, 2012, vol. 50.
17. Bittencourt, L. F., Madeira, E. R. M. HCOC: A cost optimization algorithm for workflow scheduling in hybrid clouds. Journal of Internet Services and Applications, 2011, vol. 2, 207–227.
18. Bruno, R., Ferreira, P. SCADAMAR: Scalable and Data-Efficient Internet MapReduce. Proceedings of the 2nd International Workshop on CrossCloud Systems, (CCB '14), Bordeaux, France, 2014, 2:1–2:6.
19. Buncic, P. CernVM – A Virtual Software Appliance for LHC Applications. Journal of Physics: Conference Series, 2010, 219, 042003.
20. Calheiros, R. N., Vecchiola, C., Karunamoorthy, D., Buyya, R. The Aneka platform and QoS-driven resource provisioning for elastic applications on hybrid Clouds. Future Generation Computer Systems, 2012, 28(6), 861–870, DOI: 10.1016/j.future.2011.07.005.
21. Cappello, F., Djilali, S., Fedak, G., Herault, T., Magniette, F., Néri, V., Lodygensky, O. Computing on Large-Scale Distributed Systems: Xtremweb Architecture, Programming Models, Security, Tests and Convergence with Grid. Future Generation Computer Systems, 2005, 21(3), 417–437.
22. Celesti, A., Fazio, M., Villari, M., Puliafito, A. Adding long-term availability, obfuscation, and encryption to multi-cloud storage systems, Journal of Network and Computer Applications, vol. 59, 2016, DOI: 10.1016/j.jnca.2014.09.021, 208–218.

23. Chakravarti, A., Baumgartner, G., Lauria, M. The Organic Grid: Self-Organizing Computation on a Peer-to-Peer Network. IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans, 2005, 35(3), 373–384.
24. Chen, Q., Zhang, D., Guo, M., Deng, Q., Guo, S. Samr: a self-adaptive MapReduce scheduling algorithm in heterogeneous environment. IEEE 10th International Conference on Computer and Information Technology, (CIT), 2010, 2736–2743.
25. Clifton, C. Encyclopedia Britannica. http://global.britannica.com/EBchecked/topic/1056150/data-mining. Accessed on November 7, 2017.
26. Coppa, E., Finocchi, I., On data skewness, stragglers, and MapReduce progress indicators. Proceedings of the 6th ACM Symposium on Cloud Computing, ACM, 2015, 139–152.
27. Costa, F., Silva, L., Fedak, G., Kelley, I. Optimizing Data Distribution in Desktop Grid Platforms. Parallel Processing Letters (PPL), September 2008, 18(3), 391–410.
28. Costa, F., Veiga, L., Ferreira, P. BOINC-MR: MapReduce in a Volunteer Environment. In: Meersman R. et al. (Eds.) On the Move to Meaningful Internet Systems: OTM 2012, (OTM 2012), Lecture Notes in Computer Science, 7565, Springer, Berlin, Heidelberg, 2012, 425–432.
29. Curnow, H. J., Wichmann, B. A. A Synthetic Benchmark. The Computer Journal, 1976, 19(1), 43–49.
30. Dai, W., Ibrahim, I., Bassiouni, M., An improved straggler identification scheme for data-intensive computing on cloud platforms. 4th International Conference on Cyber Security and Cloud Computing, (CSCloud), IEEE, 2017, 211–216.
31. Dean, J., Ghemawat, S. Mapreduce: Simplified Data Processing on Large Clusters. Commun, (ACM 51), January 2008, 107–113.
32. Dinda, P., O'Hallaron, D. An Extensible Toolkit for Resource Prediction in Distributed Systems. Carnegie Mellon University CMU-CS-99-138, 1999.
33. Distefano, S., Cunsolo, V. D., Puliafito, A. A Taxonomic Specification of Cloud@Home. Advanced Intelligent Computing Theories and Applications, With Aspects of Artificial Intelligence, 2010, 527–534.
34. Dong, F., Akl, S.G. Scheduling algorithms for grid computing: state of the art and open problems. Technical Report No. 2006-504, School of Computing, Queen's University, Kingston, Ontario, January 2006.

35. Duan, R., Prodan, R., Li, X. Multi-Objective Game Theoretic Schedulingof Bag-of-Tasks Workflows on Hybrid Clouds. Transactions on Cloud Computing, 2014 IEEE, vol. 2, 29–42, DOI: 10.1109/TCC.2014.2303077.
36. Fan, X., Weber, W. D., Barroso, L. A. Power Provisioning for a Warehouse-Sized Computer. In Proceedings of the 34th Annual International Symposium on Computer Architecture, (ISCA '07), ACM, New York, NY, USA, 2007, 13–23.
37. Feng, W. C., Cameron, K. W. The Green500 List: Encouraging Sustainable Supercomputing. Computer, 2007, 40(12), 50–55.
38. Frequent Itemset Mining Dataset Repository. http://fimi.ua.ac.be. Accessed on November 7, 2017.
39. Garraghan, P., Ouyang, X., Yang, R., McKee, D., Xu, J. Straggler Root-Cause and Impact Analysis for Massive-scale Virtualized Cloud Datacenters. IEEE Transactions on Services Computing, 2019, 12(1), 91–104, DOI: 10.1109/TSC.2016.2611578.
40. Gautam, J.V., Prajapati, H.B., Dabhi, V.K., Chaudhary, S. A survey on job scheduling algorithms in big data processing. In: IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT'15), Coimbatore, 2015, pp. 1–11.
41. Gill, S. S., Garraghan, P., Stankovski, V., Casale, G., Thulasiram, RK., Ghosh, S. K., Ramamohanarao, K., Buyya, R. Holistic resource management for sustainable and reliable cloud computing: an innovative solution to global challenge. Journal of Systems and Software, 2019, vol. 155, 104–129.
42. Gill, S. S., Tuli, S., Xu, M., Singh, I., Singh, K. V., Lindsay, D., Tuli, S. et al. Transformative effects of IoT, blockchain and artificial intelligence on cloud computing: evolution, vision, trends and open challenges. Internet of Things, 8:100118, 2019, DOI: 10.1016/j.iot.2019.100118.
43. Gill, S. S., Ouyang, X., Garraghan, P. Tails in the cloud: a survey and taxonomy of straggler management within large-scale cloud data centres. The Journal of Supercomputing, 2020, vol. 76, 10050–10089. DOI: 10.1007/s11227-020-03241-x.
44. Guerrero, G., Imbernón, B., Pérez-Sánchez, H., Sanz, F., Garcia, J., Cecilia, J. A Performance/Cost Evaluation for a GPU-Based Drug Discovery Application on Volunteer Computing. BioMed Research International, 2014, 474219.

45. Hamandawana, P., Mativenga, R., Kwon, S. J., Chung, T. S. EPPADS: an enhanced phase-based performance-aware dynamic scheduler for high job execution performance in large scale clusters. International Conference on Database Systems for Advanced Applications, Springer, Cham, 2019, 140–156.
46. Harutyunyan, A. CernVM Co-Pilot: A Framework for Orchestrating Virtual Machines Running Applications of LHC Experiments on the Cloud. Journal of Physics: Conference Series, 2011, 331, 062013.
47. Hashizume, K., Rosado, D., Fernandez-Medina, E., Fernandez, E. An Analysis of Security Issues for Cloud Computing. Journal of Internet Services and Applications, 2013, 4(5), 1–13.
48. Heymann, E., Senar, M., Luque, E., Livny, M., Adaptive Scheduling for Master-Worker Applications on the Computational Grid, (2000), Lect. Notes Comput. Sci. 1971. 214–227, DOI: 10.1007/3-540-44444-0_20.
49. Hindman, B., Konwinski, A., Zaharia, M., Ghodsi, A., Joseph, A.D., Katz, R.H., Shenker, S., Stoica, I. Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center. Proceedings of the 8th USENIX conference on Networked systems design and implementation, NSDI. 11, 2011, 295–308.
50. Høimyr, N. BOINC Service for Volunteer Cloud Computing. Journal of Physics: Conference Series, 396, 2012, 032057.
51. Hollingsworth, J., Maneewongvatana, S. Imprecise Calendars: An Approach to Scheduling Computational Grids. International Conference on Distributed Computing Systems, 1999.
52. Huang, X., Li, C., Luo, Y. Optimized speculative execution strategy for different workload levels in heterogeneous spark cluster. Proceedings of the 4th International Conference on Big Data and Computing, ACM, 2019, 6–10, DOI: 10.1145/3335484.3335493.
53. Yadwadkar, N. J., Ananthanarayanan., G., Katz, R., Wrangler: predictable and faster jobs using fewer resources. Proceedings of the ACM Symposium on Cloud Computing, ACM, 2014, 1–14.
54. Yoo, D., Sim, K.M., A comparative review of job scheduling for MapReduce. Cloud Computing and Intel. Syst. (CCIS), IEEE, 2011.
55. Ivashko, E., Golovin, A. Association Rules Extraction from Big Data Using BOINC-Based Enterprise Desktop Grid. Journal on Selected Topics in Nano Electronics and Computing, 2014, 2(2), 31–35.
56. Junwei, C., Stephen, J., Subhash, S., Nudd, G.R. GridFlow: Workflow management for grid computing, 198–205, DOI: 10.1109/CCGRID.2003.1199369.

57. Jurgelevičius, A., Sakalauskas, L. Big data mining using public distributed computing, Information technology and control. ISSN 1392-124X, eISSN 2335-884X, 2018, vol. 47(2), p. 236–248, DOI: 10.5755/j01.itc.47.2.19738.
58. Jurgelevičius, A., Sakalauskas, L. BOINC from the View Point of Cloud Computing, CEUR Workshop Proceedings, 1973, 2017, 61–66.
59. Jurgelevičius, A., Sakalauskas, L., Marcinkevičius, V. Task stalling for a batch of task makespan minimisation in heterogeneous multigrid computing. Computational Science and Techniques, 2021, 8, 631–638, DOI: 10.15181/csat.v8.2103.
60. Kacsuk, P., Kovacs, J., Farkas, Z., Marosi, A., Balaton, Z. Towards a Powerful European DCI Based on Desktop Grids. Journal of Grid Computing, 2011, 9, 219–239.
61. Kaklauskas, L., Sakalauskas, L., Denisovas, V. Stalling for solving slow server problem. RAIRO – Operations Research, vol. 53(4), July 2018, DOI: 10.1051/ro/2018056.
62. Krishna, L. S., Natarajan, L. P. Distributed inference with straggler mitigation. PhD dissertation, Indian institute of technology Hyderabad, 2019.
63. Lama, P., Wang, S., Zhou, X., Cheng, D. Performance isolation of data-intensive scale-out applications in a multi-tenant cloud. 2018 IEEE International Parallel and Distributed Processing Symposium, (IPDPS), Vancouver, BC, Canada, 2018, 85–94, DOI: 10.1109/IPDPS.2018.00019.
64. Lo, V., Zappala, D., Zhou, D., Liu, Y., Zhao, S. Cluster Computing on the Fly: P2p Scheduling of Idle Cycles in the Internet. In Peer-to-Peer Systems III, Springer, 2005, 227–236.
65. Lu, Y., Xu, X., Xu, J. Development of a Hybrid Manufacturing Cloud. Journal of Manufacturing Systems, 2014, 33(4), 551–566.
66. Marosi, A. C., Balaton, Z., Kacsuk, P. GenWrapper: A Generic Wrapper for Running Legacy Applications on Desktop Grids. In 3rd Workshop on Desktop Grids and Volunteer Computing Systems, (PCGrid 2009), 2009, 1–6.
67. Marosi, A. C., Kacsuk, P., Fedak, G., Lodygensky, O. Sandboxing for Desktop Grids Using Virtualization. Parallel, Distributed and Network-Based Processing, (PDP), 18th Euromicro International Conference, 2010, 559–566.

68. Marosi, A., Kovács, J., Kacsuk, P. Towards a Volunteer Cloud System. Future Generation Computer Systems, 2013, 29(6), 1442–1451.
69. McGilvary, G., Barker, A. D., Lloyd, A., Atkinson, M. V-BOINC: The Virtualization of BOINC. In 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, (CCGrid), IEEE Computer Society, 2013, 285–293.
70. McGilvary, G., Barker, A., Atkinson, M., Ad Hoc Cloud Computing: From Concept to Realization. IEEE 8th International Conference, (CLOUD 2015).
71. Mishra N., Siddiqui, S., Tripathi, J. A Compendium Over Cloud Computing Cryptographic Algorithms and Security Issues. International Journal of Information Technology, 2015, 7(1), 810–814.
72. Mohd Usama, MengchenLiu, Min Chen, Job schedulers for Big data processing in Hadoop environment: testing real-life schedulers using benchmark programs, Digital Communications and Networks, vol. 3(4), November 2017, 260–273.
73. Montes, D., Añel, J. A., Pena, T. F., Uhe, P., Wallom, D. C. H. Enabling BOINC in Infrastructure as a Service Cloud System. Geosci. Model Dev., 2017, 10, 811–826.
74. Morsky, M., Grundy, J., Müller, I. An Analysis of the Cloud Computing Security Problem. 17th Asia-Pacific Software Engineering Conference, (APSEC 2010), Sydney, Australia.
75. Nandagopal, M., Uthariaraj, V., Hierarchical Load Balancing Approach in Computational Grid Environment. International J. of Recent Trends in Engineering and Technology, vol. 3(1), May 2010.
76. Ouyang, X., Garraghan, P., Yang, R., Townend, P., Xu, J., Reducing late-timing failure at scale: Straggler root-cause analysis in cloud datacenters. Fast Abstracts in the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN, 2016.
77. Ouyang, X., Garraghan, P., McKee, D., Townend, P., Xu, J., Straggler detection in parallel computing systems through dynamic threshold calculation. 30th International Conference on Advanced Information Networking and Applications, (AINA), IEEE, 2016, 414–421.
78. Ouyang, X., Wang, C., Yang, R., Yang, G., Townend, P., Xu, J., ML-NA: a machine learning based node performance analyzer utilizing straggler statistics. 23rd International Conference on Parallel and Distributed Systems, (ICPADS), IEEE, 2017, 73–80.

79. Ozfatura, E., Gündüz, D., Ulukus, S., Speeding up distributed gradient descent by utilizing non-persistent stragglers, 2018, arXiv preprint arXiv:1808.02240.
80. Panda, B., Srinivasan, D., Ke, H., Gupta, K., Khot, V., Gunawi, H. S., IASO: a fail-slow detection and mitigation framework for distributed storage services. Proceedings of 2019 USENIX Annual Technical Conference, (ATC '19), 2019, 47–62.
81. Paranhos, D., Cirne, W., Brasileiro, F., Trading Cycles for Information: Using Replication to Schedule Bag-of-Tasks Applications on Computational Grids. Lecture Notes in Computer Science, 2790, 2002. DOI: 10.1007/978-3-540-45209-6_26.
82. Phan, T-D., Energy-efficient straggler mitigation for big data applications on the clouds. PhD dissertation, 2017, ENS Rennes.
83. Phan, T-D., Pallez, G., Ibrahim, S., Raghavan, P., A new framework for evaluating straggler detection mechanisms in MapReduce. ACM Transactions on Modeling and Performance Evaluation of Computing Systems, (TOMPECS), 2019, 4(3):14, DOI: 10.1145/3328740.
84. Poll Results: Largest Dataset Analyzed/Data Mined. https://www.kdnuggets.com/2013/04/poll-results-larg-est-dataset-analyzed-data-mined.html. Accessed on November 7, 2017.
85. Ren, X., Ananthanarayanan, G., Wierman, A., Yu, M. Hopper: decentralized speculation-aware cluster scheduling at scale. ACM SIGCOMM Computer Communication Review, ACM, 2015, 45(4), 379–392.
86. Rojko, A., Industry 4.0 Concept: Background and Overview, International Journal of Interactive Mobile Technologies (iJIM), 2017, vol. 11(5), DOI: 10.3991/ijim.v11i5.7072.
87. Running Apps in VirtualBox Virtual Machines. http://boinc.berkeley.edu/trac/wiki/VboxApps. Accessed on November 7, 2017.
88. Sahoo, J., Mohapatra, S., Lath, R. Virtualization: A Survey on Concepts, Taxonomy and Associated Security Issues. Second International Conference on Computer and Network Technology, (ICCNT), 2010, 222–226.
89. Sarmenta, L. F., Hirano, S. Bayanihan: Building and Studying Web-Based Volunteer Computing Systems Using Java. Future Generation Computer Systems, 1999, 15(5–6), 675–686.
90. Schlitter, N., Laessig, J., Fischer, S., Mierswa, I. Distributed data analytics using RapidMiner and BOINC. In: Proceedings of the 4th

RapidMinder Community Meeting and Conference, (RCOMM 2013), 2013, 81–95.
91. Segal, B., Buncic, P. LHC Cloud Computing with CernVM. PoS, (004), 2010, 28.
92. Syed, A., Mansaf, A. A Relative Study of Task Scheduling Algorithms in Cloud Computing Environment. 2nd International Conference on Contemporary Computing and Informatics, IEEE, 2016.
93. Silberstein, M., Sharov, A., Geiger, D., Schuster, A. Gridbot: Execution of Bags of Tasks in Multiple Grids. In Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, (SC '09), New York, NY, USA, 2009, 11:1–11:12.
94. Smith, J., Nair, R. VirtualMachines. Morgan Kaufmann, Elsevier, 2005.
95. Sousa, I., Queluz, M., Rodrigues, A. A Survey on QoE-oriented Wireless Resources Scheduling, Journal of Network and Computer Applications, vol. 158, 2020, DOI: 10.1016/j.jnca.2020.102594.
96. Stavrinides, G. L., Karatza, H. D. Dynamic scheduling of bags-of-tasks with sensitive input data and end-to-end deadlines in a hybrid cloud. Multimedia Tools and Applications, Springer, 2021, vol. 80, 16781–16803.
97. Sun, Y., Zhang, J., Xiong, Y., Zhu, G. Data Security and Privacy in Cloud Computing. International Journal of Distributed Sensor Networks, 2014, 10(7), 1–9.
98. Sun, P., Security and privacy protection in cloud computing: Discussions and challenges, Journal of Network and Computer Applications, 2020, DOI: 10.1016/j.jnca.2020.102642.
99. Tandon, R., Lei, Q., Dimakis, A. G., Karampatziakis, N., Gradient coding: avoiding stragglers in distributed learning. International Conference on Machine Learning, 2017, 3368–3376.
100. Tehrani, S., Shirazi, F. Factors Influencing the Adoption of Cloud Computing by Small and Medium-Sized Enterprises (SMEs). International Conference on Human Interface and the Management of Information, 2014, vol. 8522, 631–642.
101. The BOINC Wrapper. http://boinc.berkeley.edu/trac/wiki/WrapperApp. Accessed on November 7, 2017.
102. Topcuoglu, H., Hariri S., Wu M.-Y., Performance-effective and low-complexity task scheduling for heterogeneous computing. Transactions on Parallel and Distributed Systems, IEEE , 2002, 13(3), 260–274.

103. Van den Bossche, R., Vanmechelen, K., Broeckhove, J. Online cost-efficient scheduling of deadline-constrained workloads on hybrid clouds. Future Generation Computer Systems, 2013, 29(4), 973–985.
104. Vecchiola, C., Calheiros, R. N., Karunamoorthy, D., Buyya, R. Deadline-driven provisioning of resources for scientific applications in hybrid clouds with Aneka. Future Generation Computer Systems, 2012, 28(1), 58–65, DOI: 10.1016/j.future.2011.05.008.
105. Wang, B., Song, Y., Sun, Y., Liu, J. Managing Deadline-Constrained Bag-of-Tasks Jobs on Hybrid Clouds. Society for Computer Simulation International, 2016, USA.
106. Wang, D., Joshi, G., Wornell, G. Efficient task replication for fast response times in parallel computation. ACM SIGMETRICS Performance Evaluation Review, 2014, 42(1), 599–600, DOI: 10.1145/2637364.2592042.
107. Wang, D., Joshi, G., Wornell, G. Using straggler replication to reduce latency in large-scale parallel computing. ACM SIGMETRICS Performance Evaluation Review, 2015, 43(3), 7–11.
108. Weicker, R., Dhrystone, P. A Synthetic Systems Programming Benchmark. Communications of the ACM 27 (10), October 1984, 1013–1030.
109. Wisnesky, R. Evaluating Scheduling Algorithms on Distributed Computational Grids, 2010.
110. Witten, I. H., Eibe, F. Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations. Morgan Kaufmann Publishers, 2000.
111. Zaharia, M., Konwinski, A., Joseph, A. D., Katz, R. H., Stoica, I. Improving MapReduce performance in heterogeneous environments. In Proceedings of the 8th USENIX conference on Operating systems design and implementation, (OSDI'08), 2008, USA, 8(4):7, 29–42.
112. Zhang, Y., Sun, J. Novel efficient particle swarm optimization algorithms for solving QoS-demanded bag-of-tasks scheduling problems with profit maximization on hybrid clouds. Concurrency and Computation: Practice and Experience, 2017, vol. 29.
113. Zhang, Y., Zhou, J., Sun, J. Scheduling bag-of-tasks applications on hybrid clouds under due date constraints. Journal of Systems Architecture, 2019, vol. 101, 101654, DOI: 10.1016/j.sysarc.2019.101654.
114. Zhang, Y., Zhou, J., Sun, L., Mao, J., Sun, J. A Novel Firefly Algorithm for Scheduling Bag-of-Tasks Applications Under Budget

Constraints on Hybrid Clouds. IEEE Access, 2019, vol. 7, 151888-151901, DOI: 10.1109/ACCESS.2019.2948468.

UŽRAŠAMS

UŽRAŠAMS

UŽRAŠAMS

Albertas Jurgelevičius

HIBRIDINIŲ PASKIRSTYTŲ SKAIČIAVIMŲ DALIJIMOSI PLATFORMA

Daktaro disertacija
Technologijos mokslai
Informatikos inžinerija (T 007)

Redaktorė Jorūnė Rimeisytė-Nekrašienė

Vilniaus universiteto leidykla
Saulėtekio al. 9, III rūmai, LT-10222 Vilnius
El. p. info@leidykla.vu.lt, www.leidykla.vu.lt
Tiražas 20 egz.