LATVIJAS UNIVERSITĀTE

DATORIKAS FAKULTĀTE

PAŠREGULĒJOŠA AIZSARDZĪBA PRET SISTĒMAS PĀRSLODZI

MAĢISTRA DARBS

Autors: **Natālija Takere**

Stud. apl. Nr.: nl07009

Darba vadītājs: Dr.dat., prof. Jānis Bičevskis

RĪGA 2018

ANOTĀCIJA, ATSLĒGVĀRDI

Maģistra darba mērķis ir izveidot no sistēmas veiktspējas atkarīgu sistēmas pārslodzes vadības mehānismu, kas, darbojoties fonā, mēra kopējo sistēmas slodzi un balstoties uz iegūtajiem rezultātiem regulē sniedzamo servisu apjomu. Nepietiekamas veiktspējas gadījumā tiek veidotas gaidītāju rindas jaunajiem lietotājiem, tā ļaujot efektīvi pabeigt iesāktās darbības sistēmā esošajiem lietotājiem. Lietotāji, kas atrodas gaidītāju rindā, tiek informēti par aptuveno gaidīšanas laiku, pēc kura varēs piekļūt pieprasītajai funkcionalitātei. Darbā ir aprakstīts risinājums, kas veicina augstāku servisu saņemšanas kvalitāti, izvairoties no sistēmas pārslodzes radītām atteicēm un samazinot ar sistēmas lēnu darbību neapmierinātu lietotāju skaitu.

Atslēgvārdi: veiktspējas testēšana, sistēmas pārslodzes vadības mehānisms, rindu veidošanas mehānisms.

ANNOTATION, KEYWORDS

Within the Master's thesis “Self-regulating Protection Against System Overload” the aim is to create a system overload control (queuing) mechanism, which is based on performance measurements of the system. It operates in the background, measuring the total system load and regulates the amount of services provided, thus avoiding overloading of the system. In the case of inadequate performance, waiting queues are created for new users, allowing the users of the system effectively complete their actions. Users waiting in the queue are notified of the approximate waiting time after which their access will be granted. As a result, a solution is described that will promote higher quality of service, avoiding system overloads and reducing the number of disappointed users.

Keywords: performance testing, system overload control, queuing mechanism.

AUTOREFERĀTS

Darba galvenais mērķis ir izveidot no sistēmas veiktspējas atkarīgu sistēmas pārslodzes vadības mehānismu, kas ļauj pasargāt sistēmu no pārslodzes paredzēti (vai neparedzēti) liela lietotāju pieprasījumu skaita dēļ. Tādējādi izveidotais risinājums ļauj pasargāties no sistēmas atteicēm, kas ir radušās labdabības un ļaunprātīgas sistēmas ekspluatācijas rezultātā.

Autore maģistra darba ietvaros ir izstrādājusi un darbā aprakstījusi sistēmas pārslodzes vadības mehānisma prototipu, kura ietvaros tika veikta rindu mehānisma teorētiskā un praktiskā modeļa, atbilstošās datu bāzes un saistīto servisu izstrāde. Izstrādātajam prototipam tika veikti slodzes testi sadarbojoties ar 2P sistēmas izstrādātājiem. Slodzes testi pārliecināja, ka izstrādātais prototips izpilda izvirzītos mērķus.

Darba rezultātā izstrādātais prototips ļauj lietotājam brīvi izmantot sistēmas funkcionalitāti, pēc kuras tajā brīdī nav liels pieprasījums, veidojot gaidītāju rindu citiem sistēmas moduļiem, pēc kuriem izvēlētajā brīdī ir liels pieprasījumu skaits. Šāds risinājums lielas noslodzes gadījumā ļauj lietotājiem izmantot sistēmu vismaz daļēji, kas bez ieviestā pārslodzes vadības mehānisma būtu vispār nepieejama.

Darba izstrādes gaitā tika veikta arī literatūras izpēte – ilustrēts nefunkcionālo prasību neviennozīmīgums un autonomisko sistēmu izstrādes izaicinājumi. Literatūras izklāsts ir balstīts gan uz grāmatām, gan zinātniskajām publikācijām, gan citiem informācijas avotiem, kas ir norādīti izmantotās literatūras sarakstā. Problēmas aktualitātes aprakstā izmantotas atsauces uz sabiedriskajiem medijiem, kas atspoguļoja dažādu plaši izmantotu sistēmu nedienas ar pārāk lielu pieprasījumu skaitu.

SATURA RĀDĪTĀJS

APZĪMĒJUMU SARAKSTS

Apzīmējums	Skaidrojums
IS	Informāciju sistēma
HRIS	Personāla pārvaldības informāciju sistēma
QCM	Pārslodzes vadības mehānisms
API	Lietojumprogrammu saskarne
2P	2People personāla pārvaldības sistēma
DDoS	DDoS uzbrukums; izkliedētais pakalpojuma atteikums

IEVADS

Mūsdienās plaši tiek izmantota dažādu darījumu veikšanas iespēja tiešsaistē, piemēram, biļešu iegāde, elektronisku izziņu un pārskatu iegūšana no valsts iestādēm vai citiem uzņēmumiem. Pie tam, lietotāji vēlas šos pakalpojumus saņemt ātri, nevis ilgstoši gaidot uz lēnu sistēmas atbildi un, iespējams, rezultāta vietā saņemot sistēmas atteici. Līdz ar lietotāju skaita pieaugumu, arvien biežāk programmatūras gala lietotāji sastopas tieši ar sistēmas veiktspējas problēmām – pārslodzes dēļ sistēmas darbība var būt ne tikai kaitinoši lēna, bet pēkšņi var kļūt vispār nepieejama.

Aplūkojot informāciju Latvijas sabiedriskajos medijos, ir atrodami vairāki piemēri, kuros plaši izmantotu sistēmu veiktspēja nav bijusi pietiekama, lai lietotāji pilnvērtīgi spētu izmantot nepieciešamo funkcionalitāti. Dziesmu un Deju svētku ieejas biļešu tirgotāji "Biļešu paradīze" ir saņēmuši plašu kritiku par gausu un teju neiespējamu biļešu iegādi internetā uz Dziesmu un Deju svētku pasākumiem[1]. Pie tam, šāda situācija ar lēnu, apgrūtinātu un praktiski neiespējamu biļešu iegādi atkārtojas teju katrā Dziesmu un Deju svētku biļešu tirdzniecības reizē. Lietotāju neapmierinātība ir izrādīta arī Valsts ieņēmumu dienesta (VID) elektroniskās deklarēšanās sistēmai, kas, lielā lietotāju skaita pieplūduma dēļ, nedarbojās jau gada ienākumu deklarācijas iesniegšanas pirmajā dienā[2]. [1] Līdzīgas problēmas ar pieejamību šogad ir piemeklējušas arī Latvijas Republikas E-veselības sistēmu (https://eveseliba.gov.lv), kas periodiski nebija pieejama lietotājiem pārāk liela pieprasījumu skaita dēļ. [2]

Iepriekš minētie piemēri skaidri norāda, ka līdz ar pieaugošo lietotāju skaitu un to vajadzībām pēc servisu un pakalpojumu pieejamības, pieaug nefunkcionālo prasību testēšanas nozīme programmatūras izstrādes procesā. Sistēmas veiktspējas testēšana, kas klientiem bieži vien ir viens no visvieglāk saprotamajiem un ar sistēmas lietošanu saistītajiem nefunkcionālo testu veidiem, ir svarīga sistēmas funkcionēšanas pārbaudes sastāvdaļa. Šī darba ietvaros, sistēmas veiktspējas dati tiek izmantoti pašregulējoša sistēmas pārslodzes vadības mehānisma darbībai, kas ļauj klientu pieprasījumiem nepieciešamības gadījumā piemērot rindu veidošanas mehānismu.

---

[1] Kritika atspoguļota ziņu portālos, piemēram, https://www.lsm.lv/raksts/kultura/dziesmu-un-deju-svetki/bilesu-paradize-virtuala-rinda-dziesmu-svetku-bilesu-pardosana-bija-apieta.a270325/

[2] Informē sabiedriskie mediji http://www.tvnet.lv/financenet/finansu_zinas/649592-jau_pirmaja_deklaraciju_iesniegsanas_diena_nedarbojas_vid_e_sistema

Darba ietvaros izveidotais sistēmas pārslodzes mehānisms:

- Ļauj izvairīties no sistēmas pārslodzes (ievērojami samazina sistēmas atteices risku, kas varētu rasties palielinoties lietotāju pieprasījumu skaitam);
- Nepietiekamas veiktspējas gadījumā veido gaidītāju rindas jaunajiem lietotājiem, tā ļaujot efektīvi pabeigt iesākto darbību jau esošajiem aktīvajiem lietotājiem;
- Informē lietotājus par rindas veidošanos konkrētiem sistēmas moduļiem, kuru lietošanai ir nepieciešama labāka sistēmas veiktspēja;
- Informē lietotājus par aptuveno gaidīšanas laiku, pēc kura varēs piekļūt izvēlētajai sistēmas funkcionalitātei.

1. NEFUNKCIONĀLĀS PRASĪBAS PROGRAMMATŪRAS IZSTRĀDĒ

Mūsdienās kā ierasta prakse tiek uztverti mehānismi, kas pasargā elektroierīces no pārslodzes radītām sekām, piemēram, pārkaršanas. Tomēr apskatot informāciju sistēmas, ikdienā samērā bieži sastopam situācijas, kad veidojot sistēmu, nefunkcionālās prasības netiek skatītas vai arī tiek apskatītas tikai virspusīgi un nav iekļauta pat elementāra aizsardzība pret sistēmas pārslodzi. Tā var rasties gan pēkšņas lietotāju aktivitātes (piemēram, biļešu iegāde uz kādu masu apmeklētu pasākumu, atskaišu ieniegšana), gan ļaunprātīgas darbības rezultātā (piemēram, DDoS uzbrukums). Nefunkcionālo prasību detalizētāka apzināšana un sistēmas pašregulējošu elementu ieviešana var ievērojami uzlabot sistēmas lietošanas kvalitāti un pasargāt to no pārmērīga lietotāju skaita radītām pārslodzēm.

1.1. Problēmu piemēri mūsdienu sistēmās

Apskatot ikdienā izmantojamus pakalpojumus, visai ātri var atrast piemērus, kur pašregulējoša aizsardzība pret sistēmas pārslodzi būtu varējusi novērst gan sistēmas atteices, gan arī samazināt neapmierinātu lietotāju skaitu. Kā viens no piemēriem ir biļešu iegāde uz Dziesmu un Deju svētku pasākumiem. Iepriekšējās reizēs, kad tika uzsākta šo biļešu tirgošana, tiešsaistes sistēma tika pārslogota un pircējiem nebija pieejama. Saskaņā ar publiski pieejamo informāciju[3], datos, ko sniedz sabiedriskie mediji, šogad trīs stundu laikā tika pārdoti aptuveni 32000 biļešu, kas ir 178 pasākuma ieejas biļetes minūtē. Pastāv arī citi risinājumi biļešu tirdzniecības nodrošināšanai. Jaunā Rīgas teātra biļešu pārdošanas dati, ko sniedz sistēmas uzturēšanas personāls[4], liecina, ka 10 minūšu laikā tiek nopirktas apmēram 3000 biļetes (6 izrādes zālē ar apmēram 500 vietām), kas ir apmēram 300 ieejas biļetes minūtē.

[3] Informē sabiedriskie mediji https://www.lsm.lv/raksts/kultura/dziesmu-un-deju-svetki/dziesmu-svetku-drudzis-izpardotas-biletes-uz-centralajiem-notikumiem.a270060/

[4] Privāta komunikācija ar sistēmas uzturētājiem

Šogad kā risinājums Dziesmu un Deju svētku biļešu tirdzniecības sistēmas atteicēm, tika ieviests rindu mehānisms, kas brīdināja potenciālos pircējus par acīmredzamo – iespējamu gaidīšanas laiku (skat. attēlu 1.1).

1.1. att. **Rindu regulēšanas risinājums pasākumu biļešu tirdzniecības lapā**

Tomēr šis risinājums nenorādīja pat aptuveno gaidīšanas laiku (kas varēja variēt no dažām minūtēm līdz pat daudzām stundām). Pie tam, šādam risinājumam ir acīredzami arī citas nepilnības, piemēram, pārlādējot weblapu, interneta pārlūku vai datoru, saskaņā ar vietnē ievietoto informāciju, vieta rindā tika neatgriezeniski zaudēta un lietotājs ievietots virtuālajā rindā no jauna. Tiem lietotājiem, kas vēlējās šajā laikā iegādāties biļetes uz citiem pasākumiem, tas nebija pieejams, kas varēja radīt gan potenciālo pircēju pieaugošu neapmierinātību ar šo servisu, gan arī potenciālu negatīvu finansiālu ietekmi pašam biļešu tirgotājam.

Kā līdzīgus piemērus var minēt arī problēmas ar atskaišu iesniegšanu VID elektroniskās deklarēšanas sistēmā EDS (it īpaši 2017.gadā). Pirmajā gada ienākumu deklarācijas iesniegšanas dienā lietotāju aktivitāte bija pārāk liela, lai esošā sistēma spētu apkalpot visus pieprasījumus [1]. Sistēma darbojās lēni, tās funkciju izmantošana bija praktiski neiespējama un lietotāju neapmierinātība strauji auga. Arī šis gadījums, līdzīgi kā Dziesmu un Deju svētku biļešu iegādes sarežģījumi sistēmas pārslodzes dēļ, tika plaši iztirzāts presē [1]. Pašregulējošs rindu veidošanas mehānisms ir viens no risinājumu variantiem, lai regulētu lielo pieprasījuma skaita plūsmu un ļautu jau sistēmā esošiem lietotājiem efektīvi pabeigt iesāktās darbības.

Kā iespējamu ļaunprātīgu uzbrukumu piemēru var minēt Latvijas Republikas E-veselības sistēmas (https://eveseliba.gov.lv) darbības traucējumus vairāku stundu garumā [2], kas tika klasificēts kā pārslodzes uzbrukums ar mērķi padarīt nepieejamu sistēmu. Arī šajā gadījumā pašregulējoša rindu mehānisma iekļaušana daļēji palīdzētu risināt problēmu – informētu lietotājus par palēninātu sistēmas darbību un potenciālo gaidīšanas laiku, kā arī ļautu sistēmā esošajiem, aktīvajiem lietotājiem pabeigt iesākto darbību bez traucējošas sistēmas darbības palēninašanās vai tās nepieejamības vispār.

1.2. Risināmās problēmas apraksts darbā izmantotajā sistēmā

Uzņēmumā tiek izmantota personāla pārvaldības sistēma (HRIS), kas ietver dažādus funkcionālos moduļus – aptauju, atskaišu, darba plūsmas, iekšējās saziņas u.c. Pieprasījums pēc dažu moduļu izsaukumiem ir pīķveidīgs – piemēram, darbiniekiem nosūtītu anketu aizpildīšana notiek praktiski vienlaicīgi, tādējādi radot pēkšņu, īslaicīgu slodzi sistēmai. Tāpat sarežģītu atskaišu ģenerēšana katrai nodaļai tiek veikta katra mēneša pirmās darba dienas rītā. Lai gan pasūtītājs detalizētāk ir aprakstījis funkcionālās prasības, nefunkcionālās prasības praktiski neminot, tomēr ir svarīgi, lai konkrēta moduļa paaugstinātas slodzes brīžos sistēma spētu saglabāt arī pārējo funkcionalitāti (piemēram, atskaišu ģenerēšanu, iekšējo saziņu) un darbotos bez atteicēm. Lai to nodrošinātu, ir izveidots pašregulējošs veiktspējas vadības mehānisms.

Lai gan pastāv dažādi sistēmas veiktspējas vadības mehānismi, tomēr tie vai nu nav publiski pieejami citiem lietotājiem, vai arī izveidotais risinājums finansiāli ir pārāk dārgs. Tāpēc ir izveidots savs risinājums – pašregulējošs, no sistēmas veiktspējas atkarīgs rindu veidošanas mehānisms, kas sākotnēji kā prototips ir pārbaudīts HRIS.

1.3. Nefunkcionālās prasības

Programmatūras izstrādē nefunkcionālās prasības tiek lietotas, lai norādītu uz kritērijiem, kas nodrošina kopējo sistēmas darbību (ne tikai atbild par atsevišķas funkcionalitātes ieviešanu). [3, 4, 5]

Nefunkcionālās prasības bieži vien ir formātā “sistēmai ir jābūt <*prasība*>”, atšķirībā no funkcionālajām prasībām, kas norāda uz lietām, kas jāveic sistēmai. Tās mēdz saukt arī par prasībām, kas neattiecas uz sistēmas funkcionalitāti, bet uz sistēmas kvalitātes atribūtiem,

kvalitātes mērķiem un ar funkcionalitāti nesaistītām prasībām jeb cik labi sistēma dara to, kas tai jādara. [3, 6, 7]

Nefunkcionālo prasību izteikšana, lai gan bieži aizmirsta vai pārprasta, ir tikpat svarīga kā funkcionālo prasību definēšana, lai funkcionālās prasības tiktu implementētas "pareizi". Nefunkcionālās prasības bieži vien ir grūti nomērāmas, neskaidras vai pat var nebūt norādītas projekta dokumentācijā vispār. Šāda situācija var rasties, ja:

- klientam, definējot prasības, daļa no prasībām liekas pašsaprotamas, tāpēc netiek izteiktas;
- laika trūkuma dēļ prasību veidošanas vai analīzes laikā nefunkcionālajām prasībām tiek pievērsta ļoti maza uzmanība vai netiek pievērsta vispār;
- komandai pietrūkst zināšanu vai pieredzes par efektīvu nefunkcionālo prasību apstrādi dažādos programmatūras izstrādes etapos utt.

Viedās tehnoloģijas ir risinājums, kurā komponentes tiek iebūvētas programmatūrā, lai uzlabotu implementēto nefunkcionālo prasību realizācijas kvalitāti un atvieglotu šādas programmatūras uzturēšanu un atjaunināšanu. Tās risina nefunkcionālo programmatūras īpašību iekļaušanu sistēmā ne tikai ar cilvēkresursu (piemēram, programmētāju), bet arī ar pašas programmatūras palīdzību, tādējādi automātiski veicot vismaz daļēju kvalitātes aspektu nodrošināšanu. [8]

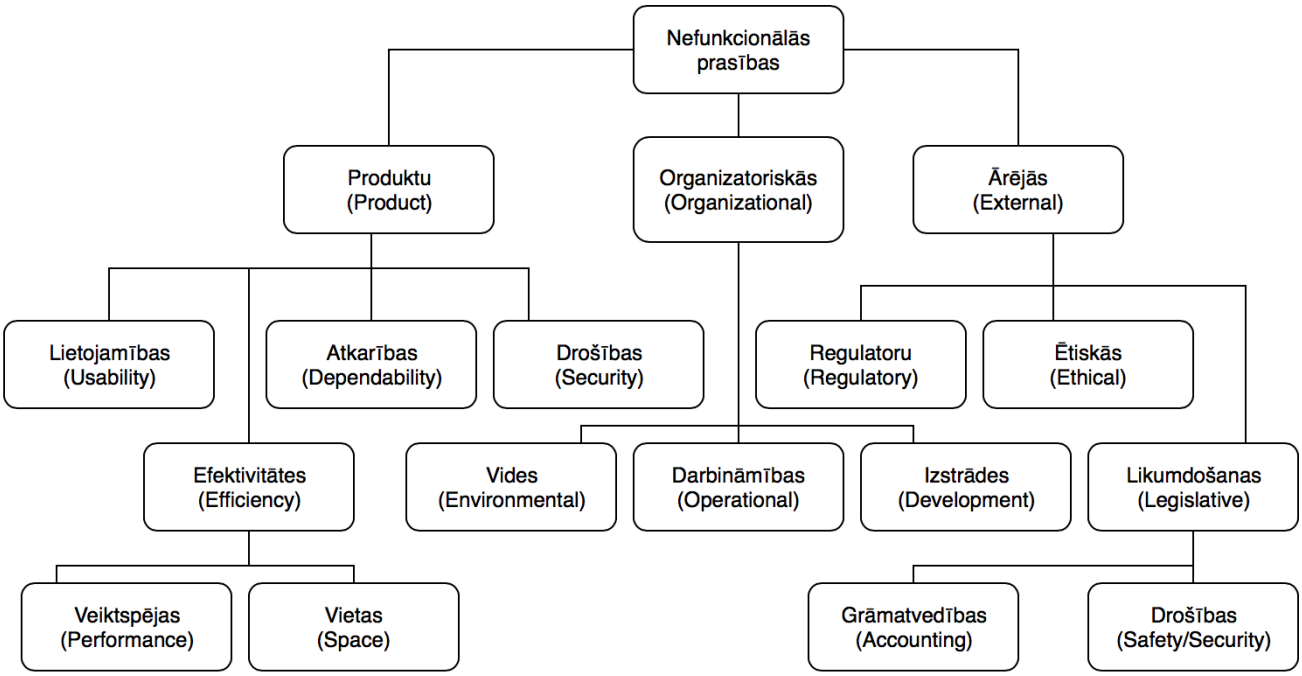
Nefunkcionālās prasības visbiežāk ir sastopamas programmatūras prasību specifikācijā (*software requirement specification*), kurā tiek aprakstīti visi izstrādājamās programmatūras aspekti. Tomēr jāņem vērā, ka ne vienmēr tiek rakstīta formāla programmatūras prasību specifikācija. Rakstītā versija biežāk tiek izmantota, ja programmatūras izstrādi veic ārējā kompānija, specifikācijas trūkums var izraisīt ievērojamas nevēlamas sekas vai arī pati sistēma ir ļoti sarežģīta. [9]

Tā kā ļoti plaša literatūras daļa ir pieejama angļu valodā, terminu tulkojumiem iekavās norādīts nosaukums oriģinālvalodā, lai izvairītos no pārpratumiem neviennozīmīgu tulkojumu gadījumā. Darbā ir sastopami arī vieni un tie paši termini ar dažādu skaidrojumu – tas ir atstāts ar nolūku parādīt, ka dažādos avotos viens un tas pats termins var tikt skaidrots atšķirīgi (atkarībā no konteksta).

1.4. Nefunkcionālo prasību iedalījums

Lai izprastu, kādēļ nefunkcionālās prasības dažkārt netiek pilnvērtīgi iekļautas prasību dokumentā, tiks apskatīta šo prasību dažādā "daba" un veidi. Literatūrā ir sastopami dažādi nefunkcionālo prasību iedalījumi, un nav vispāratzīts tikai viens, precīzi definēts nefunkcionālo prasību iedalījums. Tie var arī variēt atkarībā no sistēmas specifikas. Šeit aplūkosim dažus no iedalījumu veidiem:

1. Izpildes (*execution*) kvalitātes un attīstības (*evolution*) kvalitātes nefunkcionālās prasības [10] ,
2. Ārējās (*exterior, external*) un iekšējās (*interior, internal*) nefunkcionālās prasības. [3, 11]
3. *Wieger* piedāvātajā programmatūras prasību specifikācijā ir atsevišķi norādītas sekojošas nefunkcionālās prasības – veiktspēja (*performance*), drošība (*security*), nekaitīgums (*safety*), citi programmatūras kvalitātes atribūti, [3, 9]
4. *Sommerville* [12] nefunkcionālās prasības iedala trīs klasēs, kurai katrai vēl ir savas apakšlases (detalizēts sadalījums ir aprakstīts 1.1. attēlā):
    - produktu (*product*) – nosaka ar sistēmas darbību saistītās prasības, piemēram, cik ātri ir jābūt izpildītām operācijām, atmiņas izmantošanas prasības, kļūmju skaits (*failure rate*), drošības un lietojamības prasības;
    - organizācijas (*organizational*) – šīs prasības veido procedūras un noteikumi klienta un izstrādātāju kompānijās, piemēram, prasības izstrādāšanas procesam – kāda valoda, izstrādes vide un izstrādes process tiks lietoti;
    - ārējās (*external*) prasības – šajā klasē autors ietver visas nefunkcionālās prasības, kuras rada sistēmai un tās izstrādes videi ārēji faktori, piemēram, dažādu ārējo regulatoru noteiktās prasības, juridiskās un ētiskās prasības.



1.1. att. **Nefunkcionālo prasību iedalījums pēc *Sommerville* [12]**

Dažus no iedalījumiem apskatīsim detalizētāk. Pirmajā punktā minētās izpildes (*execution*) kvalitātes nefunkcionālās prasības sevī ietver īpašības, kas ir novērojamas programmas izpildes laikā – drošība (*security*), nekaitīgums (*safety*), lietojamība. Attīstības (*evolution*) kvalitātes nefunkcionālās prasības ir iestrādātas pašas sistēmas struktūrā, piemēram, testējamība, uzturamība un mērogojamība. [10]

Otrajā punktā minētās ārējās (*exterior, external*) nefunkcionālās prasības parasti ir nozīmīgas klientam un ir novērojamas sistēmā. Tās var ietvert [3, 11]:

- Uzticamība (*reliability*) – cik sistēma ir pieejama;
- Lietojamība (*usability*) – parāda, cik viegli ir lietot un apgūt sistēmu;
- Pieejamība (*availability*) – sistēma ir pieejama lietošanai, kad nepieciešams;
- Pareizība (*correctness*) – vai tiek izpildītas nepieciešamās lietas un tas tiek veikts pareizi (īpaši aprēķiniem);
- Izturīgums (*durability*) – laiks starp sistēmas atteicēm; var ietvert arī bojājuma apjomu sistēmas kļūdas gadījumā;
- Izskats (*appearance*) – patīkams, pievilcīgs izskats, kas var radīt papildus uzticību sistēmas darbībai;
- Nekaitīgums (*safety*) – sistēma nerada kaitējumu, bojājumus, ievainojumus;

- Drošība (*security*) – sistēma ir pieejama un lietojama tikai autorizētā veidā tikai atļautiem lietotājiem;
- Privātums (*privacy*) – privātās informācijas un telpas aizsardzība no nevēlamas piekļuves;
- Mērogojamība (*scalability*) – iespēja lietot mainīga skaita lietotājiem vai ar mainīga apjoma datiem;
- Stabilitāte (*stability*) – pakāpe, līdz kurai sistēmas īpašības un spējas nemainās;
- Integritāte (*integrity*) – datu satura un struktūras saglabāšana, it īpaši sistēmas kļūdas gadījumos;
- Lietderība (*usefulness*) – atbilstība vajadzībām;
- Iepriecinājums (*delightfulness*) – veic vairāk, nekā cerēts, radot prieku;
- Darbināmība (*operability*) – var darboties droši un efektīvi;
- Veiktspēja (*performance*) – ātrums un caurlaidspēja;
- Ietilpība (*capacity*) – lietotāju, ierakstu skaits vai datu apjoms;
- Atbalstāmība (*suportability*) – savlaicīga un noderīga palīdzība sistēmas lietošanā un darbībā;
- Pielāgojamība (*adaptability*) – spēja un ērtums, lai pielāgotu dažādiem apstākļiem;
- Izmaksu efektivitāte (*cost-effectiveness*) – iegūto papildinājumu vērtība pārsniedz to ieviešanas izmaksas.

Savukārt otrajā punktā minētās iekšējās (*interior, internal*) nefunkcionālās prasības parasti ir nozīmīgākas izstrādātājiem un uzturētājiem, nevis gala lietotājiem. Tās ietver [3, 11]:

- Efektivitāte (*efficiency*) – minimālais laiks, pūles, resursi vai izmaksas, lai izveidotu vai izmantotu risinājumu;
- Stils (*style/elegance*) – patīkams vai pārdomāts dizains un/vai tā ieviešana;
- Atkārtota izmantošana (*reusability*) – pakāpe, līdz kādai var atkārtoti izmantot starpposmu un/vai galaproduktu, nevis tos pārbūvēt;
- Struktūra (*structure*) – arhitektūras piemērotība, izturība un ekonomija;
- Datubāze (*database*) – uzglabāto datu struktūra, efektivitāte un integritāte;
- Konfigurācija (*configuration*) – citas aparatūras un programmatūras sastāvdaļas, kas ir savienotas ar sistēmu;
- Vide (*environment*) – fiziskie un tehnoloģiskie apstākļi, kurā sistēma pastāv;

- Pārnesamība (*portability*) - iespēja un ērtums izmantot risinājumu dažādās konfigurācijās vai vidēs;
- Elastīgums (*flexibility*) – iespēja un ērtums pielāgot risinājumu darbībai dažādās situācijās;
- Izsekojamība (*traceability*) – ērtums vai iespēja izsekot artefaktu cēloņus un iespējamos avotus, kas var radīt artefaktus;
- Testējamība (*testability*) – ērtums vai iespēja izveidot tādus testus, kas demonstrē, ka implementātais modelis darbojas;
- Uzturamība (*maintainability*) – produkcijas vides atjaunināšanas ātrums;
- Atbalstāmība (*supportability*) – produktu lietotāju atbalsta ātrums, uzticamība un lietderība;
- Vadāmība (*manageability*) – ērti sekot līdzi artefaktiem un to izmaiņām;
- Ražojamība (*manufacturability*) – arhitektūras īstenošanas vienkāršība;
- Darbināmība (*operability*) – sistēmas ekspluatācijas vieglums;
- Saprotamība (*understandability*) – arhitektūras vai ieviestās sistēmas arhitektūras izpratnes vienkāršība;
- Dokumentācija (*documentation*) – paskaidrojošās informācijas piemērotība un pietiekamība;
- Salāgojamība (*compatibility*) – spēja un ērtība darboties ar citiem produktiem, sistēmām un programmatūru;
- Savietojamība (*interoperability*) – spēja darboties dažādās vidēs vai konfigurācijās;
- Uzstādīšana (*installation*) – lietošanai paredzētā produkta, sistēmas vai programmatūras uzstādīšanas vienkāršība, ātrums un uzticamība;
- Lokalizācija/ internacionalizācija (*localization/internationalization*) – spēja darboties citās ģeogrāfiskajās un citas kultūras jomās;
- Izplatīšana (*distribution*) – produkta, sistēmas vai programmatūras izplatīšana lietotājiem ir vienkārša, ātra un uzticama.

Iepriekš apskatītie dažādie nefunkcionālo prasību iedalījumi parāda, cik dažādas nefunkcionālās īpašības var ietekmēt izstrādātās programmatūras kvalitāti. Tādēļ veidojot programmatūru, ir jāņem vērā nefunkcionālās prasības, kā arī to ietekme uz veidojamo gala produktu ir jāizskaidro pasūtītājam, ja vien tās nav iekļautas jau prasību sarakstā.

1.5. Nefunkcionālo prasību mērīšana

Bieži vien ir sastopama situācija, kad klienti apraksta nefunkcionālās prasības kā vispārējus mērķus, kā piemēram, vienkārši un ērti lietojama sistēma vai ātra sistēmas atbilde. Savukārt izstrādes brīdī šādi vispārēji mērķi rada plašu interpretācijas iespēju un var kļūt par strīdus objektu brīdī, kad sistēma ir piegādāta klientam. Tā kā nav iespējams objektīvi izmērīt vispārēju mērķi, ir jācenšas prasībās iekļaut kaut nelielas iespējas padarīt šo mērķi mērāmu [5]. Dažādu nefunkcionālo prasību piemēri ar mērāmiem mērķiem ir apskatāmi 1.1. tabulā. [12,13]

1.1 Tabula

Mērāmu nefunkcionālo prasību veidošana

Ātrums	Apstrādātās transakcijas sekundē Lietotāja/notikuma atbildes laiks Ekrāna atjaunošanās laiks
Izmērs	Megabaiti ROM čipu skaits
Lietošanas vienkāršība	Apmācības laiks Palīdzības ietvaru skaits sistēmā Peles klikšķu skaits
Uzticamība	Vidējais laiks līdz sistēmas kļūmei Sistēmas nepieejamības varbūtība Kļūmju biežuma rādītājs Pieejamība
Izturība	Atjaunošanās laiks pēc kļūmes Daļa no notikumiem (procentos), kas izraisa kļūmi Sabojātu datu varbūtība kļūmes gadījumā
Pārnesamība	Mērķa sistēmu skaits Moduļu skaits, kas atkarīgi no citām sistēmām Koda daļa procentos, kas nav pārnesama Sistēmu skaits, kurās programmatūra darbojas

Iekļaujot metrikas nefunkcionālo prasību formulējumā, tiek iegūta iespēja testēšanas laikā iegūt precīzāku pārskatu vai sistēma atbilst izvirzītajām prasībām [3]. Izmantotās metrikas var norādīt apgabalu (*fit criteria*), kādā ir jāiekļaujas izmērītajam atbildes rādītājam. [7, 12, 14]

Tomēr praksē klientiem bieži vien ir sarežģīti pārveidot savas vēlmes izmērāmos mērķos. Jāņem arī vērā, ka dažām prasībām, piemēram, uzturamībai, nav zināmas metrikas, ko varētu lietot. Tomēr, pat ja šādas metrikas eksistē, klienti ne vienmēr prot tās sasaistīt ar savām vēlmēm – norādītais skaitlis prasībās viņiem neizsaka, kā tas izpaužas viņu ikdienas datora lietošanas pieredzē. Līdz ar to, nefunkcionālo prasību metrikām ir jābūt ne tikai objektīvi izmērāmām, bet tās ir jāizskaidro arī klientam saprotamā valodā, ņemot vērā klienta ikdienas datora lietošanas prasmes un zināšanas. Jāņem arī vērā, ka izmaksas objektīvai nefunkcionālās prasību mērīšanai var būt visai augstas un klientam tādēļ nepieņemamas. [12, 15, 16]

Nefunkcionālo prasību definēšanas sākumā vērtību apgabali (skaitliskas sagaidāmās vērtības) var nebūt pieejami, tomēr prasībām ir jābūt identificētām un pēc iespējas arī pievienotām metrikām, kas vēlākos posmos tiks papildinātas. Ja nepieciešams, ir jānorāda arī metode, ar kādu ir jāveic mērījums. [16]

Prasību definēšanā var izmantot dažādus tirgū pieejamos risinājumus. Viens no tiem ir *Planguage* – neformāla, strukturēta, atslēgas vārdu vadīta plānošanas valoda, kura tika izveidota 1988.gadā. Izmantojot *Planguage* var izveidot detalizētas prasības, kas satur arī identifikatoru, prasības īsu apkopojumu vai atslēgas vārdu, pamatojumu, prioritāti, prasības statusu, autoru utt., lai veidotu efektīvas (mērāmas) prasības. [7]

1.6. Nefunkcionālo prasību testēšana

Nefunkcionālās prasības apraksta nevis dažādas funkcijas, bet to uzvedības vai visas kopējās sistēmas atribūtus, piemēram, cik "labi" sistēmai vai tās daļai būtu jāveic savas funkcijas. Nefunkcionālās prasības bieži vien var noteikt, cik apmierināts būs klients vai gala lietotājs ar izveidoto sistēmu un cik patīkami būs to lietot. Atbilstoši ISO 9126 standartam, šādas prasības ir lietojamība (*usability*), uzticamība (*reliability*), efektivitāte (*efficiency*). Jāņem arī vērā – jo ātrāk un vienkāršāk sistēmu var pielāgot mainītajām prasībām, jo apmierinātāki gan klienti, gan gala lietotāji. [6, 7]

Viena no biežākajām problēmām nefunkcionālo prasību testēšanā ir to neprecizitāte, nepilnīgums un grūtības pielāgot testējamajai īpašībai mērvienības (piemēram, sistēmas

lietošanas ērtums (prasība "programmai ir jābūt ērti lietojamai") ir neviennozīmīgs un var krasi atšķirties starp cilvēkiem ar dažādu datora lietošanas prasmju līmeni). Līdz ar to, nefunkcionālo prasību testēšana var būt ļoti subjektīva un plaši interpretējama. Piemēram, vārdi "ātri, vienkārši, laicīgi, bieži, intuitīvi, normāli, uzticami, draudzīgi, droši" ir plaši interpretējami. Arī nepabeigtas prasības, kas satur, piemēram, frāzes kā "vismaz; iekļaujot, bet neaprobežojoties ar; vai vēlāk; tāds kā", apgrūtina nefunkcionālo prasību testēšanu (piemēram, prasība "sistēmai jāspēj nodrošināt vismaz simts lietotāju darbību" – prasības apgabals ir nenoslēgts, šādi pieļaujot plašas interpretācijas). Sistēmtestu veicēju piedalīšanās nefunkcionālo prasību formulēšanā var ievērojami uzlabot izveidoto prasību testējamību un veidot tās vieglāk izmērāmas. Nefunkcionālajās prasības parasti tiek norādīta vēlamā mērījumu amplitūda vai skala (atšķirībā no funkcionālajām prasībām, kur rezultāts var būt arī "ir/nav"), piemēram, definējot vērtību apgabalus, kuros mērījums tiks uzskatīts vai nu par veiksmīgu vai kļūdainu. Jāņem vērā, ka daudzas nefunkcionālās prasības šķiet tik pašsaprotamas, ka netiek nemaz minētas izstrādājot sistēmas prasības. Tomēr pat šādām prasībām ir jābūt validētām, jo tās var izrādīties nozīmīgas sistēmas darbībai. [6, 7]

Saskaņā ar *Myers* [6], testējot būtu jāņem vērā sekojošas nefunkcionālās prasības:

- Veiktspējas (*performance*) tests – apstrādes ātruma un atbildes laika mērīšana noteiktiem lietošanas gadījumiem, parasti mēra kā atkarību no noslodzes;
- Slodzes (*load*) tests – sistēmas uzvedības mērīšana palielinot noslodzi (piemēram, vienlaicīgi strādājošo lietotāju vai transakciju skaits);
- Spriedzes (*stress*) tests – pārslogotas sistēmas uzvedības novērošana;
- Apjoma (*volume*) tests – sistēmas uzvedības novērošana atkarībā no datu apjoma (piemēram, ļoti lielu failu apstrāde);
- Stabilitātes (*stability*) vai uzticamības (*reliability*) pārbaude sistēmas darbības laikā (piem., vidējais laiks starp kļūmēm vai kļūmes biežums);
- Drošības (*security*) pārbaude – pret neatļautu piekļuvi, DoS (pakalpojumatteices uzbrukums) utt.
- Izturības pārbaude (*robustness*) – sistēmas reakcijas novērtēšana darbības kļūdu, nepareizas programmēšanas vai aparatūras kļūmes gadījumos, kā arī izņēmumu gadījumu apstrādes un darbības atjaunošanās pārbaude;
- Savietojamības un datu konversijas (*compatibility and data conversion*) pārbaude – savietojamības pārbaude ar citām sistēmām, datu importēšana / eksportēšana utt.;

- Lietojamības (*usability*) pārbaude, cik vienkārši ir iemācīties sistēmu, darbības efektivitātes pārbaude, sistēmas atbildes saprotamība (paturot prātā dažādu galalietotāju grupu specifiskās vajadzības);
- Dažādas sistēmas konfigurācijas pārbaude, piemēram, dažādas operētājsistēmas versijas, lietotāja saskarnes valoda, aparatūras modelis;
- Dokumentācijas pārbaude – vai tā atbilstoši realitātei ataino sistēmas uzvedību (piem., lietotāja rokasgrāmata un programmas saskarne);
- Uzturamības (*maintainabiltiy*) pārbaude – novērtē sistēmas dokumentācijas saprotamību, atbilstību ekspluatācijā esošajai sistēmas versijai; pārbauda, vai sistēmai ir modulāra struktūra utt.

Pārdomāta un laicīgi izplānota nefunkcionālo prasību testēšana ļauj pārliecinošāk pārbaudīt sistēmas atbilstību klienta vajadzībām, iekļaujot pārbaudē parametrus, kas pasūtītājam var šķist pašsaprotami. Savukārt mērvienību pielāgošana testējamajai programmatūras īpašībai ļauj atkārtot un salīdzināt šādus mērījumus, tādējādi novērtējot, vai programmatūras sniegums ir uzlabojies.

1.7. Veiktspējas testēšana - pārslodzes vadīšana

Viena no nefunkcionālajām prasībām ir sistēmas veiktspējas rādītāji. Akadēmiskajā terminu datubāzē *AkadTerm* veiktspēja raksturota kā "Datu apstrādes sistēmas spēja izpildīt paredzētās funkcijas. Kā veiktspējas kvantitatīvie raksturojumi parasti tiek izmantoti atbildes laiks, caurlaidspēja, transakciju skaits u.c. Datora veiktspējas raksturošanai izmanto, piem., izpildāmo operāciju skaitu laika vienībā." [17]

Veiktspējas testu veidi var atšķirties atkarībā no testa mērķa, testējamās sistēmas īpatnībām un klientu prasībām. Vispārīgi tiek apskatīti sekojoši veiktspējas testu veidi (skat. attēlu 1.2) [18, 19, 20]:

- Slodzes (*load*) – iespējams, viens no populārākajiem nefunkcionālo testu veidiem. Var tikt izmantots, lai noteiktu sistēmas atbildes laiku plānotajam pieprasījumu skaitam;
- Spriedzes (*stress*) – pārbauda sistēmas izturības robežu vai norāda sistēmas darbību pie ievērojami lielākas noslodzes (paralēlu pieprasījumu/lietotāju skaits, transakciju skaits noteiktā laika vienībā) nekā plānots izmantot ikdienā;

- Izturības (*soak, endurance*) – pārbauda sistēmas noturību un atmiņas izlietojumu darbojoties ilgstošā laika posmā, tiek novērots, vai pēc ilgāka laika posma transakciju izpildes laiks nepagarinās;
- Pīķa (*spike*) – pēkšņas sistēmas noslodzes palielināšanās un samazināšanās tests. Tiek vērtēts sistēmas atbildes laiks pie ievērojamas, pēkšņas, īstermiņa noslodzes maiņas;
- Konfigurācijas (*configuration*) – dažādu sistēmas iestatījumu maiņa ar mērķi atrast esošajai biznesa situācijai atbilstošāko sistēmas konfigurāciju (veiksmīgāko slodzes sadali sistēmai);
- Izolācijas (*isolation*) – tiek atkārtots tas tests, kas iepriekš izraisīja kļūmi, tādējādi atrodot kļūdaino vietu;
- Interneta (*internet*) – pakalpojumiem, kas ir paredzēti dažādu ģeogrāfisko kontinentu interneta lietotājiem, veiktspējas testi tiek veidoti radot slodzi no ģeogrāfiski dažādās vietās izvietotiem datoriem. [21]

1.2 att. **Veiktspējas testu veidi**

Veiktspējas testu rezultāti ne tikai parāda sistēmas darbības ātrumu dažādam pieprasījumu skaitam vai ļauj salīdzināt divas dažādas sistēmas pie vienādas noslodzes, bet arī var norādīt uz sistēmas moduļiem ar zemāku veiktspēju. [20, 21]

Veiktspējas testēšanai tirgū ir pieejams plašs dažādu rīku klāsts, ir pieejami gan bezmaksas, gan maksas risinājumi. Lai izvēlētos kompānijai piemērotāko rīku, vajadzīgais darba apjoms bieži vien atgāni nelielu projektu, jo sākotnēji ir jāapzinās kompānijas vajadzības un vēlmes, no piedāvātajiem rīkiem ir jāatlasa piemērotākie, kā arī jāveic izmēģinājums ar izvēlētajiem rīkiem, lai izvērtētu katra atbilstību izvēlēto problēmu risināšanai.

Iespēja kaut daļēji automatizēt nefunkcionālo prasību pārbaudi ar viedo tehnoloģiju palīdzību ir būtisks solis sistēmas kopējās kvalitātes un lietotāju apmierinātības nodrošināšanā. Veikspējas parametru (piemēram, pieprasījuma atbildes laiks, resursu izlietojums) analizēšana un izmantošana pašoptimizācijai [22] tiek izmantota arī šī darba ietvaros izveidotajā rindu mehānismā.

2. AUTONOMISKAS SISTĒMAS

Ar katru gadu informāciju tehnoloģijas attīstās aizvien vairāk, izveidotie risinājumi kļūst aizvien sarežģītāki un to pārvaldība ar cilvēkresursiem prasa aizvien vairāk laika. Pie tam, programmatūras risinājumiem arvien vairāk integrējoties cilvēku ikdienā (piemēram, "gudrās mājas" risinājumi, pašbraucoši auto), tie kļūst aizvien komplicētāki un lietotājam bez specifiskām zināšanām aizvien sarežģītāk apkalpojami. Lai informāciju sistēmu instalēšanā, konfigurēšanā, atjaunošanā un uzturēšanā izveidotu kaut daļēju no cilvēka tiešas darbības neatkarīgu risinājumu, *IBM* 2001.gadā piedāvāja savu autonomiskās skaitļošanas (*autonomic computing*) manifestu [22].

2.1. IBM manifests

Līdzīgi kā cilvēka organismā, kurā dažādas sistēmas savstarpēji sadarbojas, ir autonomas un pašregulējošas, arī IBM piedāvā vīziju, kurā autonomiskā skaitļošana ir evolucionārs process ar mērķi tuvoties pašregulējošām informāciju tehnoloģiju sistēmām. *IBM* norāda, ka autonomiskas, pašregulējošas sistēmas:

- pārzina savu struktūru,
- spēj nepārtraukti "uzlabot" sevi,
- pārvalda sistēmas sadaļas,
- spēj pielāgoties neparedzamiem apstākļiem,
- spēj novērst kļūmes un atgūties no notikušajām kļūdām, kā arī
- uzturēt drošu vidi.

Rezultātā klientiem tiek nodrošināta "elastīgāka" sistēma, kas vismaz daļēji nepieciešamos labojumus un uzturēšanas darbus veic pati. IBM norāda [22], ka autonomiskā skaitļošana palīdz risināt virkni sarežģījumu kompānijām, kas ikdienā izmanto informāciju tehnoloģijas, piemēram, gadījumus, kad programmatūras uzturēšanas izmaksas ir augstas, bet efektivitāte zema; iespējas trūkumu "nemanāmi" pārvaldīt infrastruktūru; zemu uzturēšanas ātrumu un ierobežotu informācijas tehnoloģiju elastīgumu; sarežģītu un neefektīvu heterogēnu un kompleksas vides pārvaldību un citus [23, 24].

IBM savā manifestā norāda uz četrām autonomiskas skaitļošanas pamatīpašībām (skat. attēlu 2.1), kas piemīt sistēmām, kas ir spējīgas pašas sevi vadīt (*self-management*) [23]:

- Paškonfigurācija (*Self-configuration*) – sistēma ir spējīga automātiski mainīt savus iekšējos parametrus, tādējādi pielāgojoties dinamiskām vides izmaiņām.
- Pašaizsardzība (Self-protection) – lai spētu saglabāt savu drošību un neskartību, sistēma spēj paredzēt, identificēt un novērst dažādu veidu draudus.
- Pašoptimizācija (Self-optimization) – sistēma pati optimizē savu darbību, lai, ņemot vērā nospraustos mērķus, uzlabotu savu efektivitāti.
- Pašatjaunošanās (Self-healing) – lai paaugstinātu sistēmas pieejamību, tā pati atrod, izolē un salabo bojātās komponentes.

2.1 att. **Autonomisko sistēmu pamatīpašības**

Šīs četras pamatīpašības ir paplašināti izklāstītas IBM aprakstītos astoņos raksturlielumos, kas nosaka autonomisku sistēmu [23]:

- Tā sastāv no elementiem, kas raksturo sistēmas identitāti, kā arī tā spēj saglabāt sevis "pazīšanas" spēju;
- Tā spēj sevi pārkonfigurēt, reaģējot uz (potenciāli neparedzamām) vides izmaiņām;
- Pastāvīgi cenšas optimizēt darbību, lai sasniegtu iepriekš noteiktus kritērijus;
- Spēj noteikt un atgūties no komponentu kļūmēm;

- Spēj paredzēt, noteikt un izvairīties no dažādiem apdraudējumiem, tādējādi saglabājot drošību un integritāti;
- Iegūst informāciju par darbības vidi un atbilstoši reaģē;
- Ievieš atvērtus standartus, lai varētu eksistēt heterogēnās ekosistēmās;
- Mazinot plaisu starp biznesa mērķiem un to nodrošinošajiem informāciju tehnoloģiju procesiem, padara nemanāmāku sistēmu tehnisko sarežģītību.

Tā kā no veiktspējas atkarīga rindu veidošanas mehānisma veidošana ir pašoptimizācijas piemērs, tad šī manifesta daļa tiks apskatīta sīkāk.

2.2. Pašoptimizējošas sistēmas

Sistēmās, kuras nav pašoptimizējošas, ir atrodami simtiem nelineāru, manuāli iestatāmu parametru, kuru skaitam ir tendence pieaugt ar katru jauno programmatūras versiju. Savukārt pašoptimizējošā vidē gan sistēma, gan tās komponenetes nepārtraukti meklē iespējas uzlabot savu efektivitāti un veiktspēju. Dažādas starpprogrammatūras, datu bāžu sistēmas var saturēt simtiem konfigurējamu parametru, kam ir jāuzstāda pareizās vērtības, lai sistēma darbotos, tomēr to var izdarīt tikai daži cilvēki ar specifiskām zināšanām. Šādas sistēmas bieži vien tiek integrētas ar līdzīgām, vienlīdz sarežģītām sistēmām. Līdz ar to, šāda kompleksa mehānisma gadījumā, veiktspējas konfigurēšanas darbi vienai apakšsistēmai var radīt neprognozējamu efektu visai kopējai sistēmai. [22]

Autori [22] salīdzina pašoptimizējošu sistēmu spēju mācīties līdzīgi tiem procesiem, kas notiek cilvēka organismā, kā, piemēram, smadzeņu izmaiņas mācoties, muskuļu attīstība regulāri trenējoties. Autonomiskās sistēmas nepārtraukti meklē veidus, kā uzlabot savu darbību, tādējādi uzlabojot savu veiktspēju vai samazinot izmaksas. Vīzijā par autonomiskajām sistēmām, tās pašas pārbauda, monitorē un maina savus parametrus, kā arī atrod, pārbauda un pašas uzinstalē jaunākos atjauninājumus, kas palīdz sistēmai darboties efektīvāk. [22]

Pašoptimizējoša sistēma nepārtraukti meklē iespējas savas darbības optimizēšanai. Lai uzlabotu veiktspēju, samazinātu izmaksas vai paaugstinātu servisa kvalitāti, pašoptimizācija var notikt gan proaktīvā, gan reaktīvā veidā. Piemēram, sistēma nepārtraukti reaģē uz slodzi un optimizē veiktspēju pielāgojot pildāmos uzdevumus pieejamajiem resursiem. Var rasties arī situācijas, kad optimizācijas kritēriji konfliktē savā starpā, piemēram, paaugstināta drošība sistēmā var negatīvi ietekmēt veiktspēju vai lielāks lietotāja komforts “gudrās mājas” sistēmā var

radīt lielāku elektroenerģijas patēriņu. Savstarpēji konfliktējošu kritēriju iespējamība norāda uz faktu, ka autonomiskām, pašoptimizējošām sistēmām ir jāspēj rast kompromisu veidojot optimālo sistēmas konfigurāciju. Lai gan augsta līmeņa optimizācijas gadījumā tiek pielāgoti daudzi savstarpēji atkarīgi parametri, tomēr atsevišķu elementu optimizācija vēl negarantē uzlabojumus visā sistēmā. Līdz ar to implementējot pašoptimizāciju ir jāapskata visa kopējā sistēma, ņemot vērā gan tās dažādos darbības mērķus, gan dažādos pielāgojamos parametrus. Efektīvai pašoptimizējošai sistēmai ir jāspēj laika gaitā "sevi uzlabot", apgūstot labākās iespējamās iekšējo parametru vērtību konfigurācijas, pie tam tās darbību rezultātiem ir jāsakrīt ar kopējiem augstākā līmeņa sistēmas mērķiem. [23]

Pētījumā par biznesa procesu izpildes laika verificēšanu *Odītis* [28] atzīst, ka procesa izpildes laika verificēšana biznesa procesiem bieži vien tiek aizmirsta, lai gan tā ir uzskatāma par tikpat nozīmīgu sastāvdaļu, kā pati programmatūras verificēšana. Tas parāda, ka mūsdienu sistēmām atsevišķu komponenšu pietiekama veiktspēja var nenodrošināt veiksmīgu izpildes laiku noteiktam biznesa procesam, jo to ietekmē pārējās sistēmas komponentes. Tādējādi, rindu regulēšanas mehānisms ļautu efektīvāk vadīt biznesa procesu izpildes laiku.

2.3. Paštestēšana

Darbā par viedajām tehnoloģijām *Bičevska* [8] paštestēšanu apraksta sekojoši: "Paštestēšana – programmatūras spēja pārbaudīt pašas integritāti un darbspējas, produkcijas vidē izpildot uzkrātus testpiemērus un informējot izstrādātājus par atklātajām neatbilstībām."

Paštestēšanas raksturiezīme ir tehnoloģiskas risinājums, kas paredz, ka vismaz daļa no testēšanas mehānisma tiek iebūvēta pašā testējamajā sistēmā. Tādējādi tiek iegūts risinājums, kas ļauj sistēmai notestēt pašai savu darbspējību. Tas tiek pielietots gan kā iebūvētā paštestēšana mikroshēmās (risinājumos ar potenciāli dzīvībai kritiskām sistēmām, piemēram, medicīnā, aviācijā), gan var tikt pielietots arī programmatūru testēšanai. [8, 25]

Apskatot programmatūras paštestēšanas funkciju, parasti tiek runāts par tās divām galvenajām komponentēm [8]:

- Sistēmas kritiskās funkcionalitātes testa piemēri un testa dati, kas pārbauda sistēmas galvenās sastāvdaļas, kuras ir nepieciešamas, lai lietotu programmatūru;
- Automātiskās testēšanas mehānisms – programmatūra, kas ir iebūvēta kopējā risinājumā. Tas dod iespēju salīdzināt automātiski izpildīto testu rezultātus ar

iepriekš noteiktām etalona vērtībām un izmantot to regresajā, t.i., izmainītas programmatūras atkārotā, testēšanā.

Tā kā paštestēšanas funkcija tiek iebūvēta jau pašā sistēmā, tās izstrāde bieži vien notiek kopā ar programmatūras izstrādi. Tas dod iespēju izmantot paštestēšanu programmatūras pārbaudē ne tikai testa vidē, bet arī izstrādes un produkcijas vidē (ja vien pasūtītājs pietiekami uzticas paštestēšanas "nekaitīgumam", respektīvi tam, ka paštestēšana nekādā veidā negatīvi neietekmē sistēmas darbību produkcijā). Lai nebūtu šī negatīvā ietekme, ir svarīgi, ka paštestēšanas darbības neveic ierakstus produkcijas datubāzē, bet izmanto to tikai informācijas lasīšanai, ja ir tāda nepieciešamība. [25]

Paštestēšanu visbiežāk izmanto visās programmatūras izstrādes un ekspluatācijas vidēs – gan pašā izstrādes vidē (notiek sistēmas izstrāde, kļūdu labošana, kā arī testu uzkrāšana), gan testu (vide, kas tiek izmantota akcepttestēšanai un pēc iespējas ir pietuvināta produkcijas vides apstākļiem) un produkcijas vidēs (reālā sistēmas ekspluatācijas vide, kurā darbojas sistēmas lietotāji). Tomēr, lai to varētu veiksmīgi izmantot, arī automatizētai paštestēšanai ir jābūt kārtīgi pārbaudītai, lai tās darbība neradītu papildu kļūdas un notestētu nepieciešamās sistēmas funkcijas, kas ir kritiskas sistēmas darbībai. [25]

Kad paštestēšanas funkcionalitāte ir izveidota un pārbaudīta, tā ir gatava lietošanai. Pie tam, sistēmu, kas satur iebūvētu paštestēšanas funkcionalitāti, var lietot dažādos veidos:

- Testu uzkrāšanas režīms – notiek esošo testu rediģēšana, dzēšana, kā arī jaunu testu veidošana un uzkrāšana.
- Paštestēšanas režīmā – tiek veikta paštestēšana – automatizēta uzkrāto testa piemēru izpilde.
- Sistēmas lietošanas režīmā – lietotājs izmanto sistēmas pamatfunkcionalitāti bez paštestēšanas izmantošanas.
- Demonstrācijas režīmā – uzkrātā testu kopa tiek izmantota demonstrējot sistēmas darbību, apmācot jaunus sistēmas lietotājus u.tml. gadījumos. [25]

Paštestēšanas realizācijā svarīga komanda ir testa punkts – programmēšanas valodas komanda kodā, kas tiek ievietota pirms kādas noteiktas, citas komandas izpildes (kas atbild par kritiski svarīgu funkcionalitāti) un ļauj testa izpildes laikā saglabāt informāciju par konkrētām darbībām, lauku vērtībām un programmas izpildes rezultātu. [8, 25]

2.4. Paštestēšanas ieguvumi

Tā kā sistēmu dažādība aizvien pieaug, tās var būt savstarpēji savietojamas un kļūst arvien sarežģītākas, tad nākotnē var rasties situācija, kad izveidotais risinājums būs pārāk komplicēts un plašs, lai reālā laikā, reaģējot uz klientu pieprasījumiem laicīgi, šādas sistēmas varētu instalēt, konfigurēt, uzturēt, optimizēt un apvienot [22]. Savukārt autonomisko sistēmu uzdevums ir samazināt sistēmas apkalpošanai nepieciešamo cilvēkresursu apjomu, kā arī pacelt programmatūras izstrādes lēmumu pieņemšanu uz augstāku līmeni. [8]

Ar paštestēšanu saistītos ieguvumus var iedalīt divās daļās – īstermiņa un ilgtermiņa ieguvumos. Īstermiņa ieguvumi galvenokārt ir saistīti ar sistēmas uzturēšanas izmaksu samazinājumu, kas tiek panākts samazinot nepieciešamību pēc cilvēkresursu iejaukšanās sistēmas darbībā. Tiek saistītas cerības, ka īstermiņā ar autonomiskajām sistēmām var panākt ne tikai izmaksu ietaupījumu, bet arī labāk izmantotus gan datortehnikas, gan programmatūras resursus, pozitīvāku lietotāju pieredzi (reaģēšana uz problēmām reālā laikā), augstāku drošību, pieejamību un stabilitāti pateicoties sistēmā iestrādātām pašatjaunošanās (*Self-healing*) spējām u.c. [8]

Ilgtermiņa ieguvumi saistīti ar augstāka līmeņa aspektiem problēmu risināšanā – veiksmīgas sadarbības starp dažādām struktūrām (bizness, indivīdi, organizācijas). Kā ilgtermiņa ieguvumi tiek minēta pieejamo resursu pārdale pēc prioritātes (brīvie resursi tiek novirzīti augstākas prioritātes uzdevumiem), globālu problēmu risināšana un sadarbība (sistēmas dalās ar resursiem un informāciju), tai skaitā masveida simulācijas iespējas u.c. [8]

Lai gan autonomiskās sistēmas un viens no tās realizācijas elementiem paštestēšana vismaz teorētiskā līmenī atrisina daudzas problēmas, tomēr jāņem vērā, ka ir jāvelta papildus resursi, lai izstrādātu paštestēšanu un to iekļautu sistēmas programmatūras kodā. Šāda pieeja var nebūt pieņemama īstermiņa projektos, tomēr ilgtermiņa projektos un apjomīgās sistēmās kvalitatīva paštestēšanas ieviešana viennozīmīgi ļaus samazināt laiku, kas nepieciešams sistēmas testēšanai un kļūdu novēršanai. [25]

3. PAŠREGULĒJOŠA AIZSARDZĪBA PRET PĀRSLODZI 2P SISTĒMĀ

Mūsdienās plaši tiek izmantota darījumu veikšanas iespēja tiešsaistē. Tomēr līdz ar lietotāju skaita pieaugumu arvien biežāk gala lietotāji sastopas ar sistēmas veiktspējas problēmām – pārslodzes dēļ sistēmas darbība ir ne tikai kaitinoši lēna, bet var izrādīties nepieejama pat iespēja autorizēties sistēmā.

Līdz ar pieaugošo lietotāju skaitu un to vajadzībām pieaug nefunkcionālo prasību nozīme, it īpaši sistēmas veiktspējas testēšana. Iespēja izmantot testēšanas rezultātus efektīvākas sistēmas darbības nodrošināšanai ir viens no sistēmas pārslodzes problēmas iespējamajiem risinājumiem. Nepietiekamas veiktspējas cēloņi var būt atkarīgi gan no tehnikas, gan kopējās sistēmas slodzes. Ja sistēmā tiek pildīti kādi citi prioritāri uzdevumi, tie var atņemt resursus, kas ir nepieciešami klientu pieprasījumu apstrādei tādējādi radot pat sistēmas atteices.

No sistēmas veiktspējas atkarīgā, regulējamā sistēmas funkcionalitātes bloku vadības mehānisma QCM mērķi 2P sistēmā ir sekojoši:

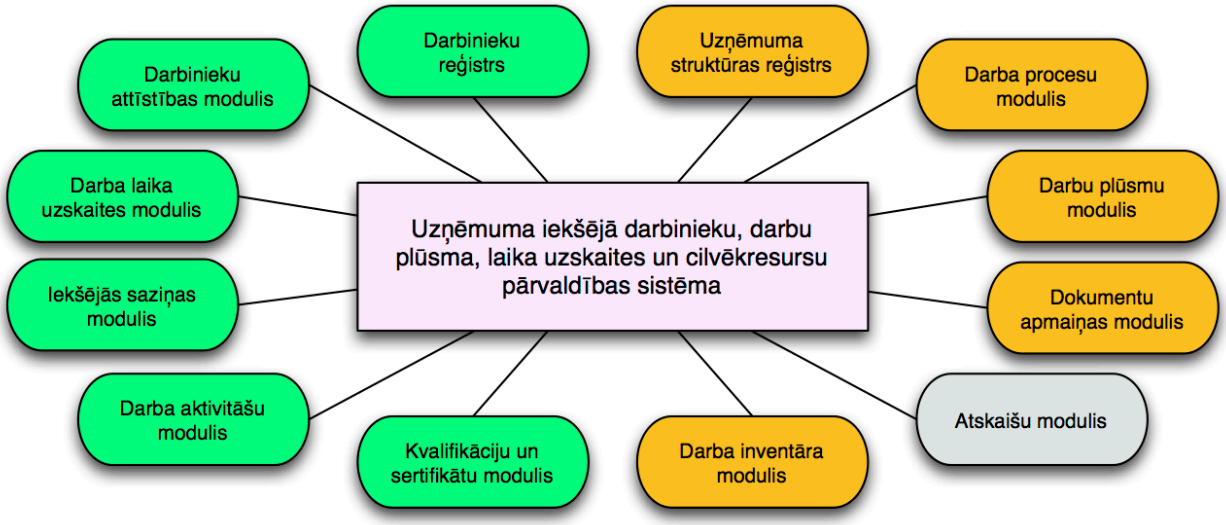
- Ļaut izvairīties no sistēmas pārslodzes (ievērojami samazināt sistēmas atteices risku, kas varētu rasties palielinoties lietotāju pieprasījumu skaitam);
- Nepietiekamas veiktspējas gadījumā veidot gaidītāju rindas jaunajiem lietotājiem, tā ļaujot efektīvi pabeigt iesākto darbību jau esošajiem aktīvajiem lietotājiem;
- Informēt lietotājus par rindas veidošanos tiem sistēmas moduļiem, kuru lietošanai ir nepieciešama labāka sistēmas veiktspēja;
- Informēt lietotājus par aptuveno gaidīšanas laiku, pēc kura varēs piekļūt izvēlētajai sistēmas funkcionalitātei.

Rezultātā ir izveidots risinājums, kas veicina augstāku servisu un pakalpojumu saņemšanas kvalitāti, izvairoties no sistēmas pārslodzes radītām atteicēm un samazinot ar sistēmas lēnu darbību neapmierināto lietotāju skaitu.

3.1. Papildināmās sistēmas un risināmo problēmu apskats

Plānotā funkcionalitāte tiek izstrādāta pilotprojekta ietvaros uzņēmuma personāla pārvaldības sistēmā (HRIS), kas satur arī iekšējās saziņas, darba plūsmas, u.c. moduļus. Šīs IS

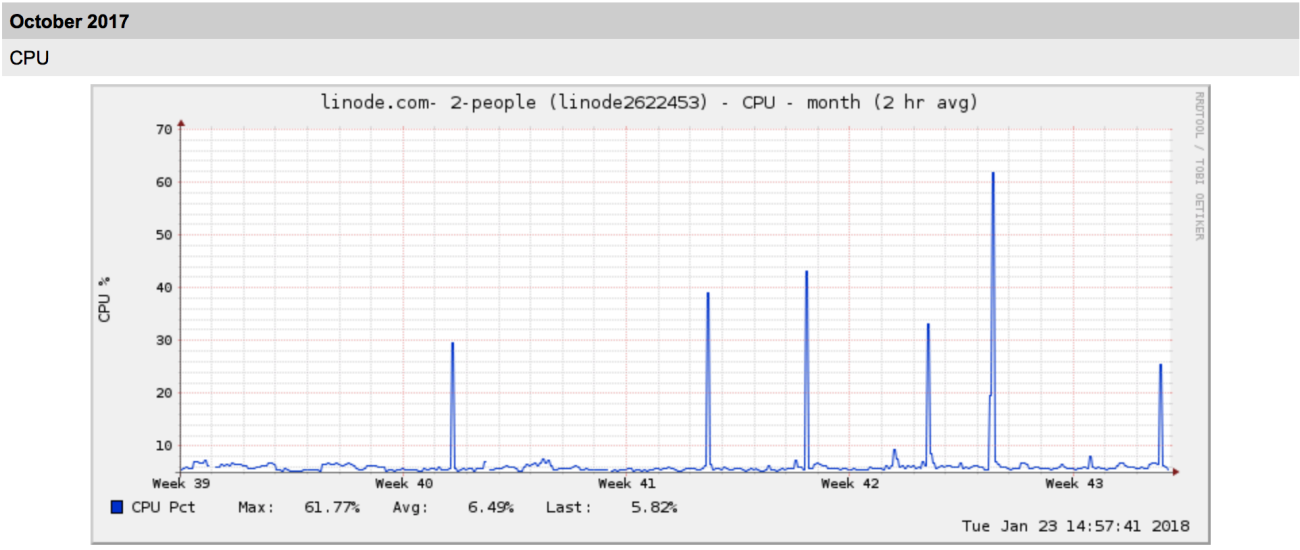
mērķis ir nodrošināt pārskatāmu un efektīvu uzņēmuma cilvēkresursu un iekšējās komunikācijas pārvaldības rīku. Risinājums tiek realizēts programmatūras kā pakalpojuma veidā.



3.1. att. **Sistēmas funkcionalitātes moduļi**

HRIS sevī ietver desmit dažādus funkcionalitātes moduļus (skat. attēlu 3.1) – ar zaļu krāsu atzīmēti moduļi ar funkcionalitāti tendētu uz darbiniekiem, oranžu krāsu – uz kompāniju, bet pelēku krāsu – uz atskaitēm tendēti moduļi. Lai gan visi moduļi varētu radīt veiktspējas problēmas, tomēr visvarbūtīgākie ir atskaišu modulis (piemēram, liela slodze noteiktās mēneša dienās, kad tiek veidotas pārskata atskaites) un darbinieku attīstības modulis (piemēram, visiem uzņēmuma darbiniekiem ir jāaizpilda noteikta aptauja). Pie tam atskaišu modulī var tikt veidotas atskaites ar reālā laika datiem arī no citiem sistēmas moduļiem.

Tiek paredzēts, ka vidējā sistēmas noslodze būs neliela, bet ar izteiktiem periodiskiem lietotāju pieprasījumiem, kas izpaudīsies kā slodzes pieaugumi pīķa veidā, kas pēc laika atkal samazināsies (attēlā 3.2 parādīta procesora noslodze serverim testa vidē, kas paaugstinās pieaugot pieprasījumu skaitam. Sagaidāms, ka produkcijas vidē būs novērojama līdzīga tendence). Līdz ar to ir svarīgi, lai šādas pēkšņas, īstermiņa slodzes maiņas neradītu HRIS atteices vai nemazinātu kopējo darbības ātrumu. Rindu veidošanas mehānisma izstrāde radīs papildu aizsardzību HRIS pret pārslodzēm konkrētos moduļos.



3.2. att. **Procesora noslodzes grafiks**

Rindu mehānisms sākotnēji likās visnepieciešamākais diviem sistēmas moduļiem – darbinieku attīstības un atskaišu ģenerēšanas blokiem. Darbinieku attīstības modulim, tā kā tas ietver iespēju veikt darbinieku aptaujas, varētu būt visvairāk paralēlo lietotāju noteiktā laika vienībā. Aptauju saturs var tikt pielāgots katras atsevišķas kompānijas mērķiem. Iegūtie rezultāti tiek apkopoti un vadītājs tos izanalizē kopā ar darbinieku, katram uzstādot mērķus un uzdevumus, kas jāpilda, un atzīmējot tos sistēmā. Darbiniekam ir iespēja sekot līdzi gan savam, gan kompānijas progresam. Šī ir viena no svarīgākajām HRIS funkcijām. Regulāru visu darbinieku aptauju rezultātā iegūtā informācija ne tikai tiek attēlota dažādos rezultātu grafikos, bet ir arī iespēja iegūt un integrēt datus no citām, ārējām sistēmām. Ielasītie dati var tikt atspoguļoti ģenerētajās ikdienas atskaitēs.

Tā kā šādas aptaujas tiek izsūtītas vienlaikus visai kompānijai, tad arī to aizpildīšana tiek veikta apmēram vienā un tajā pašā laika posmā, tādējādi radot lielu skaitu paralēlu pieslēgumu noteiktam sistēmas modulim. Šis brīdis var būt visai kritisks HRIS darbībai, tāpēc papildu rindu veidošanas funkcionalitāte palīdzētu uzlabot efektivitāti – darbinieki, kas jau būs sākuši pildīt aptauju, var to pabeigt bez traucējošas sistēmas ātrdarbības samazināšanās, savukārt pārējie ir informēti par paredzamo gaidīšanas laiku, lai pieslēgtos šim sistēmas modulim.

Rindu mehānisms ir nepieciešams arī atskaišu modulim. Izsaucot atskaišu veidošanas moduli, notiek sarežģīta lietotāja tiesību kontrole, kas ir balstīta uz lietotāja lomu sistēmā. Lietotāja tiesību kontrole tiek veikta ne tikai balstoties uz iespēju piekļūt noteiktai sadaļai, bet

tiek kontrolēta arī piekļuve specifiskiem datiem katrā no sadaļām (piemēram, darbinieks vienlaikus var būt kādas nodaļas vadītājs, bet citas nodaļas ierindas darbinieks).

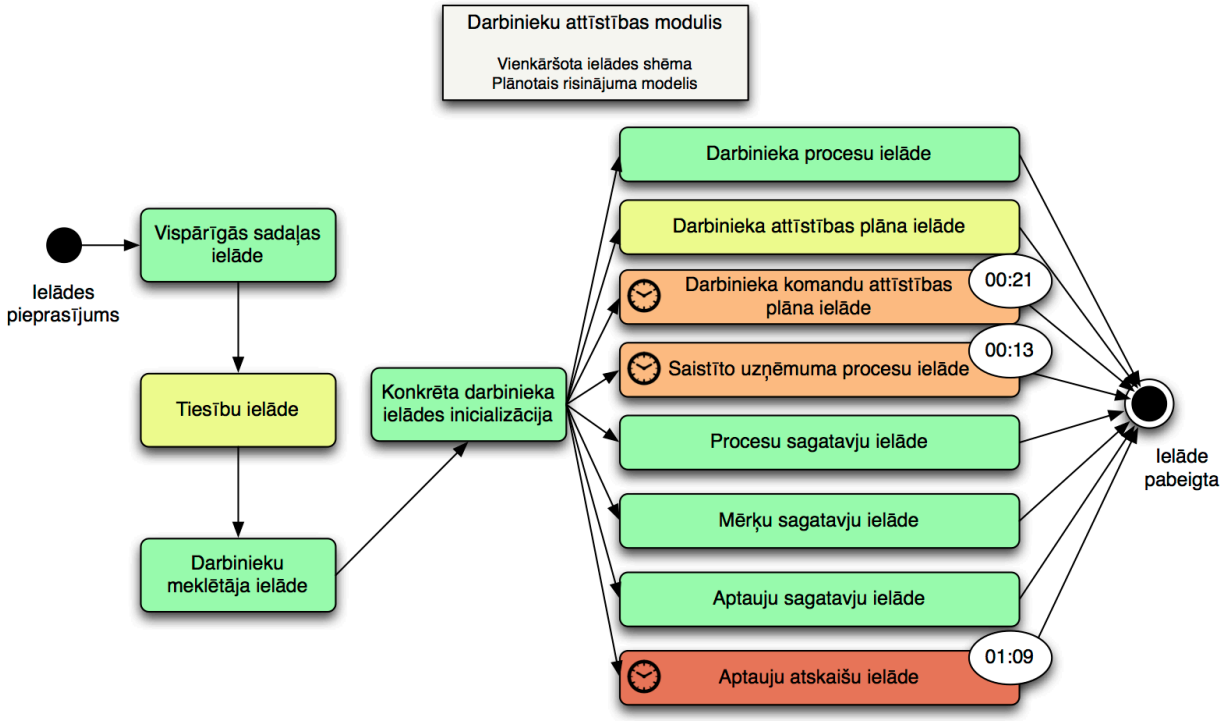
Atskaišu ģenerēšana var norisināties vienlaikus, izsaucot tās paralēlās plūsmās. Sistēmā ir arī tādi moduļi, kuri veido atskaites balstoties nevis uz vēsturiskajiem, bet aktuālajiem, laikā mainīgiem datiem, saņemot datus arī no ārējām sistēmām. Rindu mehānisma ieviešana ļautu optimāli darboties jau iesāktajiem pieprasījumiem un tikai pēc to izpildes beigām tiktu pievienoti jauni pieprasījumi, tādējādi ievērojami neietekmējot sistēmas ātrdarbību.

Lai optimizētu HRIS darbību, dažādiem sistēmas moduļiem (atšķirīgām funkcionalitātēm) ir ieviestas paralēlas rindas. Detalizētāks ieviešanas apraksts ir apskatīts darba ceturtajā nodaļā.

3.2. Risinājuma integrācija sistēmā

Maģistra darba ietvaros ir izveidots risinājums, kas analizē IS veiktspēju un, atkarībā no iegūtajiem rezultātiem, nepieciešamības gadījumā veido gaidītāju rindu uz noteiktiem (darbinieka attīstības vai atskaišu) moduļiem.

Sistēmas veiktspējas analizēšanai ir izmantota pieprasījuma izpildes laika reģistrēšana. Žurnālfaila dati tiek apkopoti speciālā rindu mehānisma datubāzē. Savāktie dati tiek analizēti un izveidotas izpildes laika normas – laika intervāli katram pieprasījumu veidam, kuros notikumi izpildās normālā, nepārslogotā sistēmas darbības laikā. Ja pieprasījuma izpildei būs nepieciešams ilgāks laiks nekā noteikts normā, šajā brīdī sāks darboties rindu veidošanas risinājums (skat. attēlu 3.3 – ar zaļo krāsu ir atzīmētas sadaļas, kas ir pieejamas uzreiz, ar oranžo – sadaļas, kuru izmantošanai būs nedaudz jāuzgaida, kamēr ar sarkano krāsu ir atzīmētas tās moduļa sadaļas, kuru piekļuvei vajadzēs visvairāk laika. Norādītais laiks ir aprēķinātais aptuvenais gaidīšanas periods, pēc kura būs pieejama vēlamā sistēmas funkcionalitāte).



3.3. att. **Plānotais risinājums darbinieku attīstības modulim**

Pieprasījuma izpildes laiku ir iespējams reģistrēt gan žurnālfailos, gan datubāzē un no tiem aprēķināt normu jeb slieksni, pēc kura pārsniegšanas savu aktīvo darbību sāks rindu veidošanas mehānisms.

3.3. Sliekšņa noteikšanas veidi

Sistēmā katram pieprasījumam ir savs optimālais izpildīšanās laiks. To var ietekmēt dažādi faktori [26]:

- Tehniskais nodrošinājums:
  - Procesora ātrums un paaudze;
  - Komunikāciju līnija datu pārvadei starp dažādām mātesplates komponentēm;
  - Operatīvās atmiņas apjoms;
  - Cietā diska apjoms un ātrums;
  - Grafiskā karte;
  - Mākoņrisinājumiem – tīkla pieejamība un noslodze.

- Paralēlo darbību skaits
    - Lietotāju veidoto pieprasījumu skaits;
    - Neracionāli SQL pieprasījumi, kas noslogo sistēmu un nav risināmi ar procesora ātruma palielināšanu.

Pieprasījuma brīdī notiekošo procesu skaits un apjoms ietekmē darbību veikšanas ātrumu. Lai noteiktu pieprasījuma apstrādes laika slieksni – pieprasījuma izpildes ilgumu sekundēs, pēc kura sāk darboties rindu veidošanas mehānisms, pieprasījumam tiek aprēķināts vidējais izpildes laiks pie normālas sistēmas noslodzes. Lai to izdarītu, tiek fiksēts laiks, kas nepieciešams no konkrētas metodes pieprasījuma izsaukuma brīža, līdz tā apstrādes beigām. Lai noteiktu pieprasījumu apstrādes laika slieksni, tiek aprēķināta konkrētas metodes apstrādes laika vidējā vērtība (saskaitīts apstrādes laika ilgums un izdalīts ar izsaukumu skaitu).

Izstrādājot informāciju sistēmas, pieprasījuma izsaukuma un izpildes laiku ir iespējams glabāt gan žurnālfailos, gan datubāzē. Datu rakstīšana žurnālfailos ir relatīvi vienkāršāk ieviešama un datu ierakstīšana ātrāka. Atšķirībā no datu ierakstīšanas datubāzē, saglabāšana datu failā nav atkarīga no papildus savienojumiem (piemēram, pieslēgums datubāzei). Tomēr teksta veidā saglabāts žurnālfails var būt grūtāk lasāms nekā datubāzes ieraksti, datus pirms apstrādes ir nepieciešams pareizi izvadīt no žurnālfaila un apstrādāt pirms tos var lietot informācijas iegūšanai. Glabājot žurnālfailus teksta veidā jāņem vērā arī citi riski – teksta faila bloķēšanās iespēja, brīvas vietas trūkums uz diska, kur tiek glabāts fails un piekļuves problēmas, kas var rasties, ja fails tiek glabāts uz servera, kuram ir neatbilstoši drošības iestatījumi. Datu saglabāšana datubāzē ļauj ērtāk atlasīt nepieciešamos datus, tie ir nepieciešamajā formātā un sagrupēti. Piekļuve datu bāzei ir relatīvi vienkāršāka nekā teksta failam un servera. Tomēr izvēloties šo risinājumu ir jāņem vērā, ka datu ierakstīšana datubāzē rada papildus slodzi, kā arī datu saglabāšanu var ietekmēt pieslēguma kvalitāte vai kļūmes datubāzē. [27]

3.4. Iespējamo sistēmas šauro vietu apskats

Sistēmā šaurās vietas iespējamas dažādos izpildes procesa soļos. Analizējot darbību plūsmu, sistēmas veiktspēju var ietekmēt:

- Lēns interneta savienojuma ātrums;
- Ierīces nepietiekama veiktspēja, no kuras tiek veidots savienojums;

- Datubāzes veiktspējas ierobežojumi (piemēram, neracionālu pieprasījumu izpilde, kas samazina pieprasījumu izpildes ātrumu);
- Palēnināta URL apstrāde – DNS (domēnu vārdu sistēma) darbība. DNS novirza URL uz nepieciešamo IP adresi (tiek secīgi pārbaudīta pārlūkprogrammas, operētājsistēmas, maršrutizātora un ISP (interneta pakalpojumu sniedzēja) kešatmiņa)
- Nepietiekama sistēmu uzturošā servera veiktspēja.

Mākoņrisinājumi ļauj dinamiski mainīt resursu pārdali, tādējādi strauja lietotāju skaita gadījumā piešķirot lielākus resursus, tomēr nespēj atrisināt pārējās iespējamās šaurās vietas sistēmā.

Esošajā Sistēmā šauro vietu veido arī dažādu pieprasījumu paralelitāte – daļā gadījumu datubāzē ir jāieraksta kopsavilkumu dati, kas ir nepieciešami nākamajām atskaitēm. Kamēr šie dati nav ierakstīti, pilnvērtīga tālāko datu analīze un atskaišu ģenerēšana nav iespējama.

4. IZSTRĀDĀTĀ RISINĀJUMA APRAKSTS

Maģistra darba ietvaros tika izstrādātais rindu mehānisma prototips, tā testēšanu veicot mākoņbāzētā uzņēmumu iekšējās HR un procesu pārvaldības sistēmas risinājumā (HRIS). Kaut arī neiedziļinoties detaļās rindu mehānisms izklausās pēc diezgan vienkāršas sistēmas funkcionalitātes, tā patieso sarežģītību iespējams izprast tikai pēc prototipa detalizētas izplānošanas un izstrādes. Tā kā darba pabeigšanas mirklī minētās sistēmas noslodze vēl nav sasniegusi tuvākajos gados plānoto, tad arī rindu mehānisma tests veikts balstoties uz testa datiem, pārbaudot to testa vidē un izmantojot testa datus.

Praktiskā piemēra izstrāde sastāvēja no 12 soļiem.

Žurnalēšanas izstrādes daļa (skat. attēlu 4.1):

1) Uzbūvēt žurnalēšanu no sistēmas žurnālfailiem teksta failos;

2) Žurnalēt sesiju, vidi, izmantoto sistēmas moduli, notikuma laiku – ielādes sākumu un beigas, statusu;

3) Uzbūvēt žurnālfailu ielasītāju, kas atmiņā ielasa ielādes informāciju.

4.1. att. **Žurnalēšanas izstrādes posmi**

Rindu mehānisma izstrādes daļa:

4) Izveidot sistēmas mehānismu, kurš ļauj definēt noslodzes parametrus;

5) Izveidot API, kas ļauj vai liedz piekļūt konkrētai funkcionalitātei;

6) Izsaukt šo API no rindu mehānismam pielāgotās sistēmas 2P;

7) Ja vidējais moduļa ielādes laiks pārsniedz noteikto slieksni, ieslēgt rindu mehānismu;

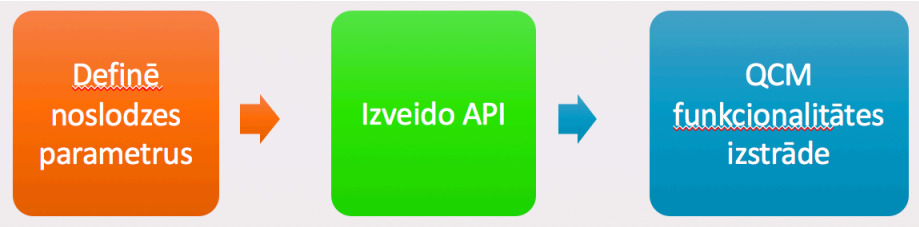
8) Atļaut piekļuvi modulim pēdējiem X lietotājiem;

9) Izpētīt, cik daudzi no pēdējiem X lietotājiem pēdējo Y min laikā piekļuva 2P sistēmai;

10) Ja ir atbrīvojušās kādas vietas, pielaist jaunus lietotājus;

11) Ja kāds lietotājs sen nav darbojies sistēmā (ir neaktīvs), izņemt to no atļauto lietotāju saraksta;

12) Ja pēc rindas ieviešanas atbildes laiks ir zem noteiktā sliekšņa noteiktā laika posmā, tad palielināt piekļuvju skaitu (pielaist jaunus lietotājus), tādā veidā nodrošinot pašregulējošu rindu veidošanas mehānismu (skat. 4.2 att.).



4.2. att. **QCM izstrādes posmi**

Turpmāk tiks apskatīts katrs no veiktajiem soļiem, vienlaikus izskaidrojot, kāds risinājums katrā no soļiem tika izvēlēts, pamatojot konkrētās izvēles priekšrocības un aprakstot arī to trūkumus.

Kā jau tika minēts iepriekš, ir iespējams veikt sistēmas datu žurnalēšanu dažādos veidos. Vienkāršākais un, iespējams, populārākais veids ir žurnāla reģistrēšana speciālos žurnālu failos. Retāk, tomēr visai plaši, tiek izmantota arī iespēja reģistrēt žurnalēšanas notikumus datubāzē. Darba praktiskās daļas izstrādes ietvaros tika izvērtēti katra veida risinājuma ieguvumi un trūkumi, pie tam tie tika apskatīti kontekstā ar datu turpmākās izmantošanas iespējām izstrādājot rindu veidošanas mehānismu. Galvenie jautājumi, uz kuriem tika meklētas atbildes žurnalēšanas izstrādes laikā, bija:

- Kādu žurnalēšanas datu saglabāšanas veidu izvēlēties – datubāzē vai žurnālu failos?
- Izvēloties datu rakstīšanu žurnālu failos, vai risinājumam veidot speciālus, atsevišķus rindu mehānisma žurnālu failus, vai "pa tiešo" izmantot sistēmas ģenerētos?
- Izvēloties žurnalēšanas datus saglabāt datubāzē, vai izmantot to pašu datubāzi vai veidot atsevišķu, speciāli rindu mehānismam paredzētu datubāzi?
- Ja tiek izvēlēti specifiski žurnālu faili vai atsevišķa datubāze, tad kur šos failus vai datubāzi izvietot? Uz tā paša servera vai citur?
- Kā izvēlētais risinājums ietekmēs sistēmas ātrdarbību?

- Cik ērta būs rindu mehānisma ieviešana un uzturēšana katrā no izvēlētajiem veidiem?
- Kā izvēlētais risinājums ietekmēs iespējas šo pašu risinājumu pielietot citām informāciju sistēmām?
- Kāds būtu nepieciešamās informācijas minimums, ko nepieciešams žurnalēt?

Plānojot implementāciju risinājumam ir daudz jautājumu, kurus nepieciešams izvērtēt. Reālajā darbībā, jau veidojot rindu veidošanas mehānismu praktiski, katrā atsevišķā gadījumā rodas vēl papildu jautājumi, kas ir konkrētajai situācijai specifiski. Jau risinājuma izstrādes sākuma fāzē ir skaidrs, ka kaut nedaudz mainoties sistēmas specifikai var mainīties arī optimālais risinājums, kā arī var rasties arī citi jautājumi, kas var ietekmēt risinājuma efektivitāti un problēmu risinājuma veida izvēli.

4.1. Žurnalēšanas veida izvēle

Izvēloties žurnalēšanas veidu ir skaidrs, ka abām apskatītajām metodēm – rakstīšanai žurnālu failos vai datubāzē – katrai ir savas priekšrocības un trūkumi. Abos gadījumos svarīgākais ir nodrošināt pēc iespējas ērtāku datu pieejamību un izmantošanas iespēju, tajā pašā laikā neatstājot ievērojamu iespaidu uz sistēmas ātrdarbību. Šajā posmā ir jāizvērtē arī iespējamā kļūdu negatīvā ietekme uz rindu mehānismu, kuru var radīt sistēmas tehniskās kļūdas citos sistēmas moduļos.

Pēc prototipam piesaistītās personāla pārvaldības informācijas sistēmas specifikas izpētes tika secināts, ka konkrētajā gadījumā rakstīšana žurnālu failos būtu sistēmai drošāks risinājums. Izmantotā sistēma ir būvēta kā modulāra sistēma, kuras katrs modulis slēdzas klāt vienai un tai pašai datubāzei. Līdz ar to, ja tiktu izvēlēts datubāzu risinājums, funkcionalitātes nodrošināšanai būtu jāizstrādā vai nu paralēli pieslēgumi dažādām datubāzēm, vai arī jāveido papildu pieslēgums sistēmas galvenajai datubāzei. Gan paralēlu pieslēgumu, gan tās pašas datubāzes lietošanas gadījumā rodas lieka papildu slodze, kas minētajā gadījumā nav nepieciešama un var radīt negatīvu ietekmi uz sistēmas ātrdarbību. Pie tam, arī nepieciešamais rakstāmā koda apjoms datubāzes izvēles gadījumā ir lielāks, tas iekļauj papildu pieslēguma konfigurācijas veidošanu un uzturēšanu, kas rezultātā var novest pie lielākas tehnisko kļūdu iespējamības.

Izvēloties informācijas glabāšanas risinājumu žurnālfailos, iespējams sistēmai paralēlā plūsmā likt veikt ierakstu papildu žurnāla failā. No tehniskā viedokļa tas ir vienkāršs kods, pie

tam, plānojot, ka sistēma var izaugt ļoti liela, šo funkcionalitāti vienmēr varēs atdalīt uz atsevišķiem procesoriem un atsevišķiem diskiem tādējādi minimāli palielinot slodzi galvenajai sistēmai. Rakstīšana teksta žurnālu failā ir plaši izmantota tik pat kā katrā informāciju sistēmā, it īpaši salīdzinot ar rakstīšanu vairākās, iespējams pat dažādās datubāzēs. Tas arī ļauj rindu mehānismu veidot pēc iespējams neatkarīgu no galvenās sistēmas un tajā izmantotajām tehnoloģijām tādējādi paaugstinot risinājuma atkal izmantošanas iespējas citos gadījumiem.

4.2. Failu novietojuma definēšana

Gan datubāzes, gan žurnālu failu gadījumā nepieciešams izvērtēt, kuros failos vai kurā datubāzē veikt ierakstus. Kaut arī tehniski vieglāk no galvenās informāciju sistēmas izstrādātāju puses parasti ir izmantot esošo datubāzi un esošos žurnalēšanas failus, tomēr jāņem vērā dažādi blakusefekti, kas rodas esošo elementu izmantošanas laikā. Tā kā risinājuma mērķis ir paaugstināt ātrdarbību un pieejamību esošai informāciju sistēmai, tad nepieciešams izvairīties no jau tā noslogotu komponenšu papildu noslogošanas. Salīdzinājumam, ja rakstīšana notiek neatkarīgos žurnālu failos vai neatkarīgā datubāzē, tad arī pašu rindu mehānisma komponenti iespējams izstrādāt kā neatkarīgu un integrējamu sistēmu, kuras darbināšana atstās niecīgu iespaidu un galvenās sistēmas ātrdarbību.

Līdzīga situācija kā rakstīšanas gadījumā ir arī failu lasīšanai. Ja faili vai datubāze būs novietoti neatkarīgā IT infrastruktūrā un, ja faili vai datubāze saturēs tikai nozīmīgo informāciju rindu mehānisma funkcionēšanai, tad ātrdarbības jomā netiks traucēta ne galvenās sistēmas, ne rindu mehānismu nodrošinošās komponentes darbība. Protams, rindu mehānisma komponentes izvietošana uz tā paša servera, kur galvenā sistēma, pārsvarā neradīs galvenās sistēmas ātrdarbību ietekmējošas sekas, tomēr gadījumos, kad šāda ietekme tiks novērota, iespēja ātri un vienkārši atdalīt galveno sistēmu no rindu mehānisma komponentes var kļūt ļoti nozīmīga.

Ņemot vērā šajā un iepriekšējā nodaļā aprakstītos apsvērumus un izvērtējot potenciālo kļūdu iespējamību katrā no tām, tika izlemts izvēlēties rakstīšanu atsevišķos, rindu mehānisma nodrošināšanai paredzētos žurnālu failos.

4.3. Žurnalējamā informācija

Kaut arī testam izmantotās sistēmas gadījumā tika izmantota speciāli šim mērķim radīta žurnalēšanas funkcionalitāte un fails, sistēmas universālas pielāgošanas mērķiem vēlams apsvērt iespēju izmantot vai vismaz definēt prasības arī jau eksistējošu sistēmu žurnālu failiem. Šādu failu ielasīšana nebūtu tik efektīva kā speciāli veidotu, tomēr tā ļautu atsevišķās situācijās pielāgot risinājumu, neiejaucoties vai mazāk iejaucoties galvenās sistēmas darbībā.

Lai nodrošinātu rindu mehānisma darbību, ir nepieciešams saprast, kāds ir minimālais datu apjoms, kas nepieciešams veiksmīgai rindu mehānisma ieviešanai informāciju sistēmā. Maģistra darba ietvaros izstrādātā rindu mehānisma ietvaros galvenajā sistēmā tika izveidoti žurnālu faili, kas satur šādu informāciju:

- Sistēmas vide;
- Kompānijas ID;
- Lietotāja ID;
- Izsauktais modulis;
- Izsaukuma sākuma laiks;
- Izsaukuma beigu laiks;
- Apstrādes laiks milisekundēs;
- Lietotāja rindā atrašanās stāvoklis – atzīme, vai lietotājs jau lieto izsaukto moduli vai arī joprojām tiek gaidīta piekļuve.

No augstāk uzskaitītajiem žurnālfaila parametriem (žurnālfaila piemēru skat. 4.3 att. un 1.pielikumā), svarīgākie analizējamie dati ir konkrētu lietotāju pieprasījumu apstrādes laiks.

```
"local","587765e8a1cb6f37f7772ca2","58c815beb4b65c010748628e","/units","Thu May 03 2018 15:30:36 GMT+0000 (UTC)","Thu May 03 2018 15:30:37 GMT+0000 (UTC)","387","Waiting"
"local","587765e8a1cb6f37f7772ca2","59e5b12dc204c20019b3a219","/units","Thu May 03 2018 15:30:36 GMT+0000 (UTC)","Thu May 03 2018 15:30:37 GMT+0000 (UTC)","395","Waiting"
"local","587765e8a1cb6f37f7772ca2","58d3cdf58fe51b027c86b6c5","/units","Thu May 03 2018 15:30:36 GMT+0000 (UTC)","Thu May 03 2018 15:30:37 GMT+0000 (UTC)","400","Waiting"
"local","587765e8a1cb6f37f7772ca2","58c81c78b4b65c0107486290","/units","Thu May 03 2018 15:30:36 GMT+0000 (UTC)","Thu May 03 2018 15:30:37 GMT+0000 (UTC)","584","Completed"
"local","587765e8a1cb6f37f7772ca2","587894045feedf0019e4212a","/units","Thu May 03 2018 15:30:36 GMT+0000 (UTC)","Thu May 03 2018 15:30:37 GMT+0000 (UTC)","601","Completed"
"local","587765e8a1cb6f37f7772ca2","58c7eb0c7adc7400fc5af0fe","/units","Thu May 03 2018 15:30:36 GMT+0000 (UTC)","Thu May 03 2018 15:30:37 GMT+0000 (UTC)","591","Completed"
"local","587765e8a1cb6f37f7772ca2","58c02d0437a4e40049a457e3","/units","Thu May 03 2018 15:30:36 GMT+0000 (UTC)","Thu May 03 2018 15:30:37 GMT+0000 (UTC)","596","Completed"
"local","587765e8a1cb6f37f7772ca2","59e5b12dc204c20019b3a219","/units","Thu May 03 2018 15:30:36 GMT+0000 (UTC)","Thu May 03 2018 15:30:37 GMT+0000 (UTC)","430","Waiting"
"local","587765e8a1cb6f37f7772ca2","58c81f28b4b65c0107486291","/units","Thu May 03 2018 15:30:36 GMT+0000 (UTC)","Thu May 03 2018 15:30:37 GMT+0000 (UTC)","436","Waiting"
```

4.3. att. **Žurnālfaila piemērs**

Tas attiecīgi liek žurnalēt katra lietotāja ID un pieprasījumam iztērēto laika apjomu. Ļoti nozīmīga ir arī izsaukumu sākuma laika žurnalēšana, lai varētu salīdzināt aktuālos datus ar vēsturiskajiem. No pārējiem žurnālu failu datiem pieminēšanas vērts ir izsauktais modulis, kas ļauj salīdzināt konkrētu moduļu izpildes laikus. Galvenais potenciālais ieguvums ir iespēja ieviest

rindu mehānismu ne tikai visas informāciju sistēmas līmenī, bet gan konkrētiem moduļiem. Kā piemēru var minēt 2018. gada Dziesmusvētku biļešu tirdzniecības gadījumu, kad biļešu tirgotāja mājas lapas apmeklētājiem bija jāgaida rindā ne tikai uz Dziesmu svētku, bet arī pārējo minētā servisa tirgoto pasākumu biļešu iegādi. Konkrētajā gadījumā, ja rindu mehānisms būtu nodalīts Dziesmusvētku sadaļai, tad būtu bijis iespējams bez papildu gaidīšanas iegādāties pārējās biļetes. Tas arī ir iemesls, kāpēc testa cilvēkresursu sistēmas HRIS gadījumā tiek žurnalēts izsauktais modulis – lai izvērtētu dažādu moduļu ielādes laiku un varētu pielāgot rindu mehānismu atsevišķiem moduļiem.

Nenoliedzami, ka arī cita, papildu informāciju, var būt noderīga piekļuves algoritma izstrādes ietvaros. Vienlaikus jāatceras, ka aprakstītā struktūra ļauj vienkāršotā veidā risināt dažādus integrācijas jautājumus, kas citos risinājumos varētu būt daudz laikietilpīgāki un "smagnējāki".

4.4. Žurnālu failu ielasīšana

Maģistra darba ietvaros žurnālu failu ielasīšana tika izstrādāta izmantojot PHP programmēšanas valodā uzrakstītu servisu, kas žurnālu failu saturu ieraksta MySQL datubāzē turpmākai datu analizēšanai. Tā kā žurnālu failu saturs sistēmas lietošanas laikā nemitīgi palielinās, tad tika izvēlēts risinājums, kas katru minūti veido jaunu žurnālu failu. Tādējādi faila ielasīšana un satura analīze notiek ar dažu desmitu sekunžu kavēšanos, kas rindu mehānisma nodrošināšanas gadījumā ir pieļaujams. Galvenais ieguvums šādam risinājumam ir vienkāršota failu ielasīšanas procedūra, ielasot katru failu rindu mehānisma analīzes datubāzē tikai vienu reizi un nodrošinoties pret paralēlu konkrētā faila izmantošanu, kas atsevišķos gadījumos var radīt dažādas nevēlamas sistēmas kļūdas. Ja rastos nepieciešamība, intervālu no izvēlētajām sešdesmit sekundēm elementāri varētu nomainīt uz mazāku skaitu sekunžu. Šādas laika intervāla izmaiņas ļautu vēl īsākā laikā secināt, ka nepieciešams slēgt piekļuvi daļai lietotāju, tomēr konkrētajā gadījumā, izvērtējot biznesa prasības, tika nolemts, ka pilnīgi pietiekošs ir sešdesmit sekunžu intervāls. Risinājuma kods ir sīkāk aplūkojams 2. pielikumā.

4.5. Sistēmas parametri

Jau izstrādes sākuma posmā bija skaidrs, ka rindu mehānisma darbībai ir jābūt ar iespēju konfigurēt noteiktus parametrus. Lai gan sākotnēji tika paredzēts iekļaut informāciju par sistēmai paralēli pielaižamo maksimālo lietotāju skaitu, tomēr izstrādes laikā tika saprasts, ka galvenais ir norādīt konkrētās datu apstrādes procedūras izpildes laiku. Tādējādi izpildās mērķis, kas katrā no rindu mehānismu gadījumiem ir kopīgs – nodrošināt efektīvu sistēmas darbību ar pieņemamu sistēmas atbildes laiku, kurā lietotājs saņem gaidīto rezultātu. Līdz ar to nav svarīgi ierobežot lietotāju skaitu, ja lietotāji saņem gaidīto atbildi pieņemamā laikā.

Lai šādu risinājumu padarītu optimālu un vēl drošāku, ir nepieciešams definēt arī minimālo paralēlo lietotāju skaitu, kuriem ļaut piekļuvi sistēmai. Pretējā gadījumā tiek riskēts ar situāciju, kurā dažādu tehnisku iemeslu dēļ, piemēram, interneta problēmu dēļ, sistēma var sākt atbildēt lēni uz lietotāju pieprasījumiem, savukārt rindu mehānisms neļauj nevienam lietotājam piekļūt sistēmas funkcionalitātei. Minimālā lietotāju skaita definēšana dod iespēju automātiski regulēt pielaižamo lietotāju skaitu, izvairoties no šādu problēmu radītām sekām. Vienlaikus jāatceras, ka minimālā lietotāju skaita gadījumā nedrīkstētu tikt pārsniegts pieļaujamais sistēmas atbildes laiks – ja tāda situācija tomēr rastos, tad būtu vēlams iestrādāt tūlītēju brīdinājuma nosūtīšanu sistēmas uzturētājiem, kam atbilstoši būtu jāsāk izmeklēt radušās situācijas cēloņus.

Izstrādājot sistēmas dažādos režīmus un nosakot robežu, pie kuras apstrāde ir pieņemami ātra vai pārāk lēna, nākas saskarties ar situāciju, kur vienas un tās pašas sistēmas ietvaros dažādiem funkcionalitātes blokiem nepieciešami atšķirīgi rindā iekļaušanas momenta sliekšņi. Piemēram, testam izmantotās sistēmas HRIS 2P gadījumā, lietotājam būtu nepieņemami gaidīt ilgāk par desmit sekundēm, kamēr tiktu ielādēts lietotāja profils. Vienlaikus lietotājs samierinātos ar desmit un vairāk sekunžu gaidīšanu, ja tiktu izsaukta komplicētas atskaites ģenerēšanas funkcionalitāte. No tā izriet, ka rindu mehānisma parametrus nepieciešams uzstādīt atsevišķi katram no mehānismā iekļautajiem funkcionalitātes blokiem, nevis kopīgus parametrus visiem moduļiem. Līdz ar to, lai nodrošinātu aprakstīto sistēmas darbību, tika izlemts izveidot elementāru sistēmas iestādījumu reģistru, kurā katram no sistēmas blokiem tiek norādīts gan maksimālais pieļaujamais izpildes laiks, gan minimālais lietotāju skaits.

4.6. Rindu mehānismu atbalstošais API

Lai rindu mehānisms būtu izmantojams, nepieciešams izveidot to atbalstošo programmatūras sarunāšanās iespēju – lietojumprogrammas saskarni (API). API funkcionalitāte ir aprakstāma ļoti vienkāršā veidā – kā ievadparametrus tā saņem sistēmas moduļa nosaukumu un lietotāja identifikatoru vai autentifikācijas marķiera (*token*) vērtību. Savukārt, kā atbilde tiek atgriezts statusa kods (iespējamas divas vērtības):

- 200. statuss, kurš atļauj turpināt izmantot sistēmu, vai arī
- 500. statuss, kurš norāda, ka lietotājs ir iekļauts rindā, bet pagaidām izvēlētais sistēmas modulis lietotājam nav pieejams.

Tomēr jāņem vērā, ka, lai API izpilde būtu efektīva, izstrādes procesā nedrīkst to sarežģīt. Nav pieļaujama gadījuma iestāšanās, kurā API izsaukumi aizņem pārāk ilgu laiku, tādējādi panākot, ka tieši API izraisa sistēmas ātrdarbības problēmas, nevis pašas sistēmas sarežģītā funkcionalitāte. Tomēr šādas situācijas ir samērā viegli risināmas – piemēram, ir iespējams izveidot vairākas API instances, atdalot tās atsevišķi pieprasītākajiem sistēmas moduļiem.

API ātrdarbības nodrošināšanā noteikti jāpievērš uzmanība tam, kur API tiek izvietots – neatkarīgi no izvēlētā risinājuma, 2P sistēmas galvenajam serverim jāspēj pēc iespējas efektīvāk komunicēt ar API. Viens no novietojuma veidiem ir izvietot API uz tā paša servera, kur atrodas pārējā sistēma, izdalot tam rezervētus servera resursus. Cits, un, iespējams, efektīvāks veids ir izvietot API vienotā infrastruktūrā un tīklā ar kopējo sistēmu, tomēr uz atsevišķiem serveriem. Tādējādi tiktu nodalīta funkcionalitāte un resursu izmantošana, saglabājot vieglu servisa sasniedzamību.

API funkcionalitāte, kas nodrošina rindu veidošanas mehānismu ir samērā vienkārša. Pašreizējā risinājumā API galvenā funkcija ir pārbaudīt, vai konkrētajam lietotājam var atļaut piekļūt pieprasītajai funkcionalitātei. Ja izvēlētās funkcionalitātes apstrādes laiks nepārsniedz noteikto slieksni un izvēlētais modulis ir pieejams, API par to sniedz pozitīvo atbildi. Pretējā gadījumā lietotājs tiek piereģistrēts gaidītāju rindā (ja vien lietotājs tajā nav jau ievietots). Pārējās funkcijas veic žurnālfailu analizēšanas programmatūra, kas tiks apskatīta turpmākajās nodaļās.

4.7. Rindu mehānismu uzturošā programmatūra

Rindu mehānisma darbību uztur atsevišķs, maģistra darba ietvaros izstrādātas programmatūras prototips. Minētā programmatūra veic šādas funkcijas:

- Žurnālfailu ielasīšana;
- Rindu mehānisma iestatījumu pārvaldība;
- Katra funkcionalitātes bloka rindas uzturēšana;
- Katra funkcionalitātes bloka lietojošo lietotāju uzturēšana;
- Katra funkcionalitātes bloka pašreizējās situācijas analīze;
- Katra funkcionalitātes bloka rindu izstāvējušo lietotāju piekļuves regulēšana;
- Prioritāro lietotāju apkalpošana;
- Katra funkcionalitātes bloka automātiska lietotāju skaita regulēšana.

Žurnālfailu ielasīšana jau tika apskatīta iepriekšējās nodaļās. Attiecībā uz rindu mehānisma iestatījumu pārvaldību, papildus jau aprakstītajai funkcionalitātei, programmatūrā tika iestrādāta iespēja ieslēgt vai izslēgt visu rindu mehānismu. Šī iespēja ir nepieciešama testēšanas mērķiem, kā arī, kā ātrs un ērts veids efektīvai problēmu novēršanai sistēmas kļūdu gadījumā.

4.8. Lietotāju identificēšana

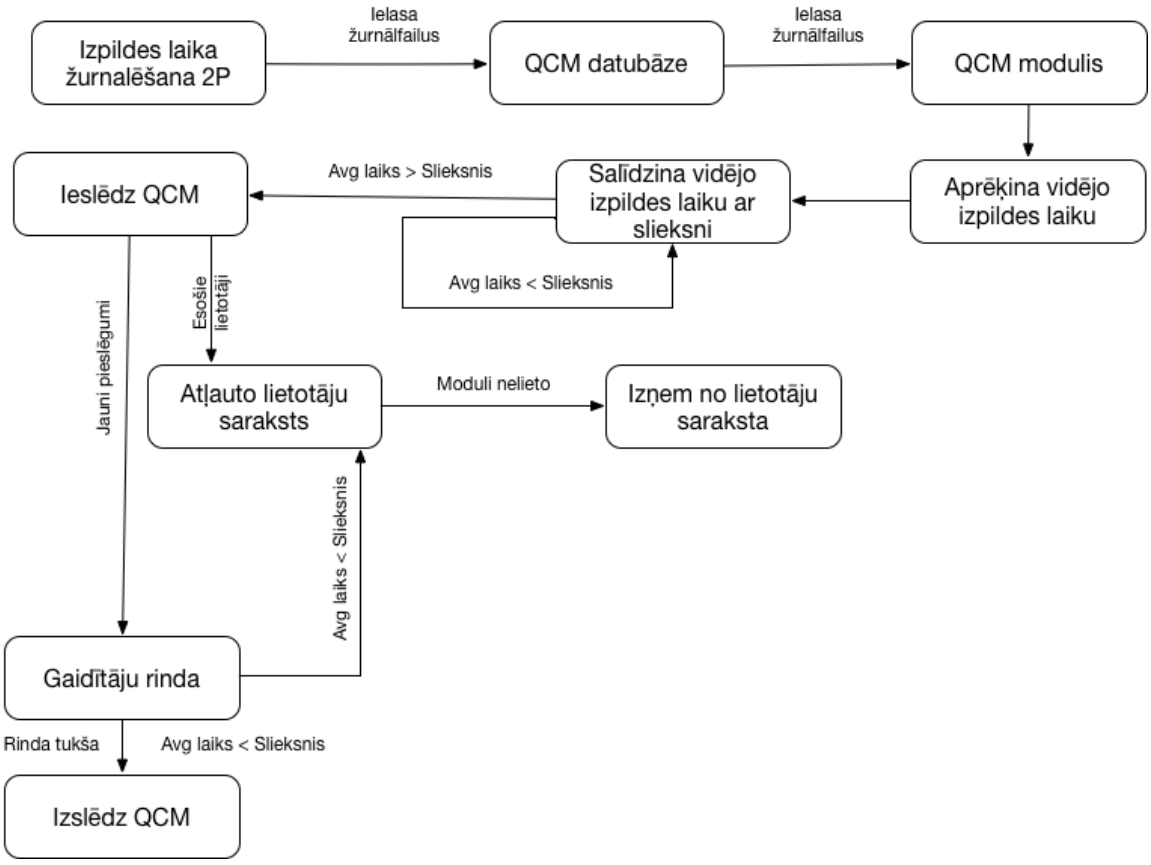
Galvenā sistēmas funkcija, bez kuras pašai rindas regulēšanas sistēmai vispār nebūtu jēgas, ir rindā esošo lietotāju pārvaldība. Tā kā maģistra darba ietvaros izstrādātās sistēmas mērķis ir nodrošināt rindu mehānismu ne tikai vienai konkrētai sistēmai, bet arī izstrādāt universālu, dažādām situācijām pielāgojamu risinājumu, tad ļoti svarīgi bija izveidot sistēmu, kura nedublētu datus no galvenās sistēmas vai darītu to pēc iespējas mazāk. Tas attiecas arī uz sistēmas galvenā objekta – tās lietotāju datu – dublēšanu.

Rezultātā tika izveidots risinājums, kurā nav nekādas nepieciešamības dublēt lietotāju datus. Atkarībā no sistēmas, rindu mehānismu nodrošinošā programmatūra lietotājus atpazīst vai nu pēc lietotāja identifikatora (ID), vai nu autentifikācijas marķiera vērtības vai arī pēc sistēmas apmeklējuma laikā piešķirtā sesijas ID. Tā kā šis sesijas vai lietotāja ID ir unikāls, ar minēto informāciju ir pilnīgi pietiekoši, lai katram rindā gaidītājam piešķirtu savu rindas numuru un pēc atkārtotiem pieprasījumiem informētu, vai lietotājs ir rindā, kurā vietā rindā viņš ir, vai viņam

piešķirt piekļuvi izsauktajam sistēmas modulim. Tādējādi rindu mehānisma programmatūrai nav zināmi detalizēti dati par lietotāju, bet tik un tā ir pietiekoši daudz informācijas katrā pieprasījumā, lai nodrošinātu vēlamo funkcionalitāti.

4.9. Sistēmas darbība

Turpinājumā tiks izskaidrots rindu mehānisma programmatūras darbības algoritms, kas attēlots 4.4 attēlā. Uzreiz pēc rindu mehānisma iedarbināšanas sāk darboties jau iepriekš detalizēti aprakstītais žurnālu failu ielasīšanas algoritms. Tomēr reālā darbībā var rasties dažādi iemesli, kāpēc žurnālu failu ielasīšana var neizdoties. Līdz ar to, ja žurnālu fails kādu sistēmā definētu laika posmu nav ielasīts, tad sistēma nosūta par to informāciju sistēmas uzturētājiem, kam tiek dots noteikts laiks problēmas izpētei un atrisināšanai. Ja problēma netiek atrisināta arī pēc šāda paziņojuma izsūtīšanas, tad, pēc noteiktā laika termiņa beigām, piekļuve sistēmai tiek atvērta visiem lietotājiem, tādējādi automātiski uz laiku atslēdzot rindu mehānisma darbību.



4.4. att. **QCM darbības algoritms**

Pēc veiksmīgas žurnālu failu ielasīšanas, kas notiek vidēji reizi minūtē, iespējams sākt analizēt pēdējo žurnālfailu saturošos datus. Galvenais analizējamais datu kopums ir izpildes laiks katram no sistēmas rindu mehānismam pielāgotajiem moduļiem. Datu atlasi iespējams veikt dažādos veidos iegūstot vidējo nepieciešamo laiku moduļa izpildei. Galvenie ar datu atlasi saistītie jautājumi, kurus šajā situācijā nepieciešams atbildēt, ir:

- Kāds ir vidējais pieprasītā moduļa izpildes laiks?
- Kāds ir vidējais pieprasītā moduļa izpildes laiks dažādos laika intervālos?
- Vai pēdējos laika intervālos pieprasītā moduļa izpildes laiks ir samazinājies?
- Vai pēdējos laika intervālos pieprasītā moduļa izpildes laiks ir mazāks par iestatījumos definēto?
- Vai mērīt tikai pieprasītā moduļa vidējo izpildes laiku, vai arī visas sistēmas kopējo?

Minēto jautājumu atbildēšana un izpildes laika salīdzināšana ar iestatījumos norādīto ļauj noteikt, vai minētajam modulim pienākusi nepieciešamība ieslēgt rindu mehānismu. Kamēr visi parametri ir zemāki par definēto konkrētā moduļa normu, tikmēr rindu mehānismam jāturpina darboties tikai situācijas analīzes režīmā, nebloķējot nevienu no pieprasījumiem. Savukārt līdz ko pieaugošais izpildes laiks pārsniedz iestatījumos definēto, jāsāk analizēt situāciju, lai noteiktu, vai jāsāk ievietot jaunos lietotājus gaidītāju sarakstā. Šāda veida analizēšana tūlītēja rindu mehānisma ieslēgšanas vietā nepieciešama, lai samazinātu varbūtību dažādiem izņēmuma gadījumiem izraisīt rindu mehānisma neparedzētu ieslēgšanu. Maģistra darba ietvaros šī analizēšana tika iekļauta vienkāršotā veidā, pārliecinoties, ka konkrētajā laikā ir bijis pietiekoši daudz pieprasījumu, kuru izpildes laiks pārsniedz definēto.

4.10. Rindu mehānisma ieslēgšana

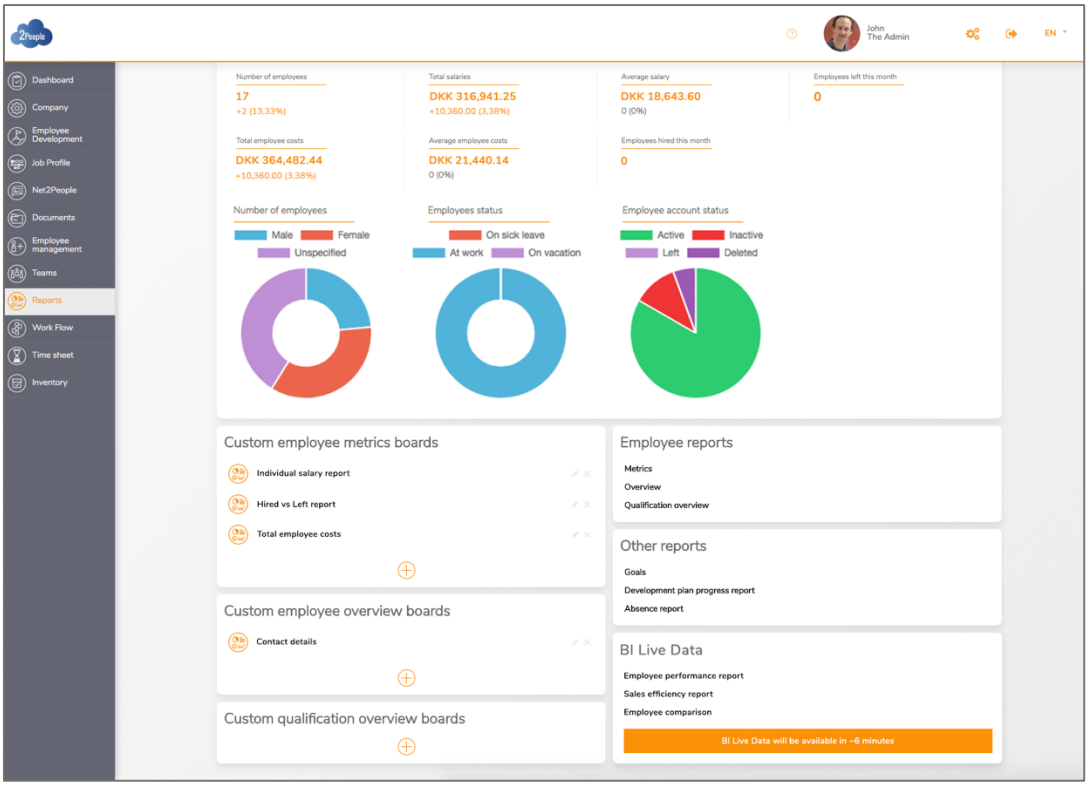
Kad rindu mehānisms ir ieslēgts, darbību uzsāk sistēmas modulis, kas nosaka, kuriem lietotājiem ļaut un kuriem liegt pašlaik piekļuvi pieprasītajam funkcionalitātes blokam. Sākotnēji tiek atlasīti visi tie lietotāji (citos gadījumos tās būtu lietotāju sesijas), kuri piekļuvuši sistēmai pēdējo dažu minūšu laikā. Lai nesabojātu lietotāju iesākto darbu, šie visi lietotāji tiek iekļauti sarakstā, kuriem ir tiesības piekļūt sistēmai. Noprotams, ka, ja jau šis lietotāju skaits iepriekš, pirms dažām minūtēm bija izraisījis pārāk lēnu sistēmas darbību, tad arī pēc visu lietotāju

iekļaušanas atļauto lietotāju sarakstā problēmai vajadzētu saglabāties. Tomēr šim argumentam ir vismaz divi pretargumenti:

- Sistēma ir veidota, lai to paralēli efektīvi varētu lietot samērā liels lietotāju skaits. Ļoti netipiski būtu, ka visi lietotāji sistēmā ieradušies vienā minūtē – visticamāk slodze tomēr ir sasniegta pakāpeniski. Līdz ar to daļa no lietotājiem savu darbu būs pabeiguši un pametīs sistēmu tuvākajā laikā, tādējādi samazinot sistēmas noslodzi.
- Mirklis, kad ieslēgt rindu mehānismu ir definējams katram modulim atsevišķi. Līdz ar to iespējams slieksni definēt nedaudz zemāku – piemēram – astoņas sekundes, divpadsmit sekunžu vietā. Tādējādi iespējams pielāgot reālo lietojamību, nesabojājot lietotāja pieredzi atļautajiem lietotājiem.

4.11. Gaidītāju rindas funkcionalitāte

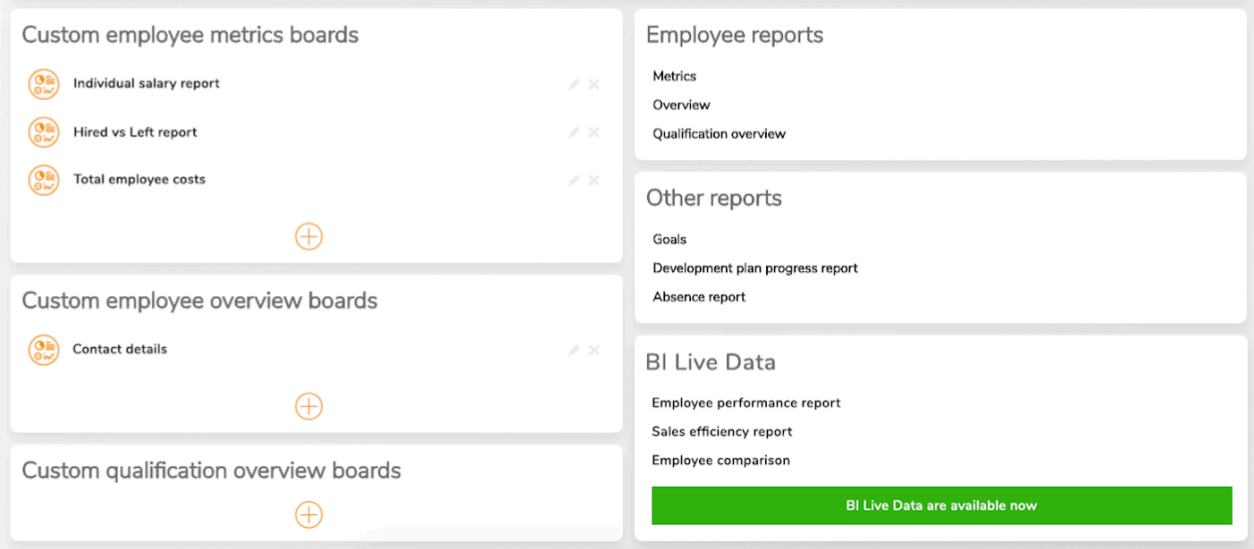
Pārējie lietotāji, kas no jauna vēlas piekļūt sistēmai, tiek visi ievietoti gaidītāju rindā. Lietotājam uz ekrāna attēlojas aptuvenais gaidīšanas laiks līdz brīdim, kad būs pieejama izvēlētā sistēmas funkcionalitāte (skat. attēlu 4.5, labais apakšējais stūris, ziņojums par moduļa *BI Live Data* pieejamību pēc aptuveni 6 minūtēm "*BI Live Data will be available in ~6 minutes*").



4.5. att. **Sistēmas modulis ar aktīvu gaidītāju rindu**

Gaidītāju rindā katram lietotājam tiek piešķirts viņa kārtas numurs (tabulas ID), ko teorētiski var nākotnē izmantot, lai parādītu aptuveno gaidīšanas laiku. Atšķirībā no risinājumiem, kuros lietotājiem neļauj pārlādēt interneta vietni, jo tādējādi tiktu pazaudēta vieta gaidītāju rindā, šajā gadījumā pārlādēšanu ir atļauts veikt, kamēr vien nemainās lietotāja ID vai sesijas numurs.

Reizi minūtē, pēc aktuālākā žurnāla faila ielasīšanas, notiek esošās situācijas pārrēķināšana, kuras rezultātā tiek izlemts, kuriem no rindā gaidošajiem lietotājiem dot piekļuvi sistēmai (skat. att. 4.6) un kuriem no piekļuvi saņēmušajiem to turpmāk liegt. Attiecībā uz piekļuves liegšanu risinājums ir diezgan vienkāršs – ir iespējams izmērīt, kuri no atļautajiem lietotājiem nav izmantojuši sistēmu pēdējo, piemēram, divdesmit minūšu laikā, un izņemt tos ārā no saraksta, kurš definē tiesības piekļūt konkrētajiem sistēmas moduļiem. Šādi rīkojoties iespējams nodrošināties, lai gadījumos, ja šie lietotāji atgriežas sistēmā un ja tādu lietotāju ir daudz, tie jūtami nepalēninātu sistēmas kopīgo darbības ātrumu.



4.6. att. **Lietotājs ar piešķirtu piekļuvi izvēlētā sistēmas moduļa izmantošanai**

Situācija ar piekļuves piešķiršanu ir nedaudz sarežģītāka. Maģistra darba ietvaros, izstrādājot sistēmas prototipu, tika izveidots vienkāršākais risinājums, kas noteiktam skaitam no rindā gaidošajiem lietotājiem piešķīra tiesības piekļūt sistēmai. Šādu risinājumu implementējot reālajā sistēmā tai nepieciešams reizi minūtē pārjautāt statusu, lai noskaidrotu, vai vēl aizvien jāgaida rindā, vai arī piekļuve ir piešķirta. Izvēlētais risinājums ir pietiekošs, lai pārliecinātos par koncepta darbību. Bez papildu uzlabojumiem var iztikt arī situācijās, kad rindu mehānisma regulētā funkcionalitāte ir ļoti sarežģīta un lietotāju skaits, kas vienlaikus mēģina lietot minēto

sistēmu, nav mērāms tūkstošos. Tādā gadījumā daži simti pieprasījumu minūtē elementāram servisam problēmas neizraisītu.

Gadījumos, kad paralēlo lietotāju skaits mērāms ar piecu un vairāk ciparu skaitļiem, risinājumu būtu vēlams papildināt ar čatu programmatūrās izmantotajiem ligzdu (*network socket*) risinājumiem. Šādā situācijā rindu mehānisma programmatūra varētu informēt galveno sistēmu, ka piekļuve ir piešķirta un nebūtu nepieciešams veikt regulāru sistēmas pārlādi, lai pārbaudītu esošo rindas stāvokli.

4.12. Prioritārā piekļuve un prioritārās rindas

Līdzīgi kā citās jomās, arī ieviešot rindu mehānismu jāņem vērā, ka ne visu lietotāju pieprasījumi ir ar identisku prioritāti. Tā kā rindu mehānisma sistēma kontrolē piekļuvi citai sistēmai, kas ietekmē dažāda ietekmes līmeņa darbinieku dažādas biznesa intereses, tad regulāri gadās gadījumi, kuros būs noderīgi ieviest arī rindu prioritātes. Kaut arī sākotnēji var uzskatīt, ka prioritāro rindu ieviest nav nepieciešams, jo tā vietā var vienkārši ļaut darboties konkrētiem lietotājiem pilnībā apejot rindu, tomēr būvējot mākoņpakalpojuma sistēmu jāparedz, ka var būt vairāki prioritārie lietotāji un sarežģītas, laikietilpīgas aprēķinu procedūras, kas neļaus efektīvi lietot sistēmu visiem prioritārajiem lietotājiem vienlaikus. Līdz ar to īpaši izveidota prioritārās sistēmas izstrāde ir efektīvs risinājums minētajā situācijā.

Maģistra darba ietvaros izstrādātajā prototipa sistēmā prioritārā rinda tika izveidota esošās standarta rindas ietvaros. Arī šajā situācijā tika veidots universāls risinājums, cenšoties neglabāt klientu datus rindu mehānismu nodrošinošās sistēmas pusē. Tā vietā jau izsaucošā sistēma veic lietotāja pozīcijas kompānijas struktūrā analīzi, lai noteiktu, vai lietotājam pienākas kāda īpaša prioritāte. Ja tāda pienākas, par to tiek informēta rindu mehānisma komponente, kas konkrētam lietotājam piekļuvi izsniegs ātrāk kā citiem.

Otra situācija, kurā prioritātes ir vēl svarīgākas, ir gadījumos, kuros lietotājs rindu ir izstāvējis, bet nav sācis lietot sistēmas funkcionalitāti. Ja, piemēram, lietotājs ir gaidījis piekļuvi ļoti noslogotam modulim stundas garumā, tad devies turpmākās trīsdesmit minūtes prom no datora un nav varējis lietot sistēmu, bet tieši šajā laikā jau ir iztecējušas visas divdesmit no viņa sesijas minūtēm, tad šāda rindu mehānisma izmantošana diezgan neilgā laikā var izraisīt patiesu lietotāju neapmierinātību. Lai šādas situācijas novērstu, izmantojamas sistēmā ieviestās prioritātes. Visiem lietotājiem, izņemot īpašos gadījumus, pirmo reizi nonākot rindā tiek piešķirta

nulles prioritāte. Gadījumā, ja lietotājs izstāv rindu, bet nesāk lietot sistēmu tam atvēlētajā laikā, tad viņam rezervētā vieta tiek atbrīvota citiem lietotājiem. Vienlaikus, lai lietotājam nākamajā piegājienā tās pašas dienas ietvaros nebūtu jāgaida visu rindu, viņa prioritāte tiek palielināta par 1, tādējādi automātiski nonākot priekšā citiem rindā esošajiem lietotājiem un jūtami samazinot kopīgo gaidīšanas laiku.

4.13. Viena vai vairākas paralēlas rindas

Viens no jautājumiem, kuru jāizlemj sistēmā iestrādājot rindu mehānismu, ir cik daudz rindu veidot. Vai nepieciešams veidot rindu sistēmai kopumā, vai arī rindu taisīt tikai konkrētai sistēmas funkcionalitātei. Lai izlemtu minēto jautājumu, nepieciešams aplūkot sistēmas biznesa specifiku kā arī izvērtēt, cik neatkarīgi ir minētās sistēmas moduļi. Maģistra darba ietvaros izmantotās testa sistēmas gadījumā izstrāde ir veikta izmantojot mikro servisu arhitektūru. Pēc savas būtības mikro servisi ļauj sistēmu veidot modulāru un sadalītu vairākās mazās sistēmās atšķirībā no monolītām jeb viengabalainām sistēmām. Tas, savukārt, ļauj ticēt, ka vairāku paralēlu rindu ieviešana varētu būt optimāls risinājums šajā situācijā. Diemžēl ne vienmēr teorētiskais risinājums ir ērti izmantojams reālā situācijā. Bieži vien integrētai mikro servisu arhitektūrā bāzētai sistēmai nākas izmantot informāciju no dažādiem servisiem vienlaikus. Tādā gadījumā ir bezjēdzīgi vai pat visu funkcionalitāti bojājoši taisīt atsevišķu rindu katram no servisiem, ņemot vērā, ka vienas rindas izstāvēšana vēl nedod iespēju pilnvērtīgi izmantot vēlamo sistēmas funkcionalitāti.

Ņemot vērā ieguvumus un trūkumus katrā no situācijām, nepieciešams izvēlēties saprātīgāko kompromisa risinājumu. Ir vēlams sadalīt rindu mehānismu vairākās paralēlās rindās, ja rindas var nodalīt citu no citas tā, ka pēc rindas pieejamā funkcionalitāte nepārklājas. Mirklī, kad funkcionalitāte, kas pieejama pēc rindas izstāvēšanas cenšas piekļūt citai funkcionalitātei, kuru regulē jau cita rinda, nepieciešams abas rindas apvienot vienā un kopīgā kontrolējošā rindā.

4.14. Automātiska lietotāju skaita regulēšana

Viens no jautājumiem, kuru nepieciešams atbildēt ieviešot rindu mehānisma komponenti, ir, cik lielam skaitam lietotāju dot piekļuvi sistēmai, kad lietotājus pārvieto no gaidītāju rindas uz atļauto lietotāju sarakstu. Šim jautājumam jeb problēmai ir vismaz divi risinājumi – maģistra

darba ietvaros tie tiks saukti kā daļēji automātiska un pilnībā automatizēta atļauto piekļuvju regulēšana.

Daļēji automatizētas piekļuves regulēšanas gadījumā katram neatkarīgajam funkcionalitātes blokam tiek definēts paralēlais atļauto lietotāju skaits. Laiku pa laikam, secinot, ka kāds no lietotājiem sistēmu vairs neizmanto, tiek atbrīvota vieta citiem, rindā gaidošajiem lietotājiem. Šāds risinājums ir vienkāršāks, tomēr pastāv augstāks risks, ka nebūs izdevies korekti noteikt vienlaikus pieļaujamo lietotāju skaitu. Līdz ar to pastāv iespēja, ka vai nu sistēma ar visu rindu mehānismu būs lēna, vai arī tieši pretējais – tā darbosies ātri, rindā bezjēdzīgi gaidot daudziem lietotājiem. Būtībā sistēmas kapacitāte atļautu lielāku lietotāju skaitu, kas neietekmētu citam cita sistēmas lietojamību, tomēr tā kā paralēlo lietotāju skaits ir fiksēts, tad daļai lietotāju nākas bezjēdzīgi gaidīt rindā.

Pilnībā automatizētas piekļuves regulēšanas gadījumā nav nepieciešams definēt konkrētu paralēlo lietotāju skaitu, kuriem piešķirt piekļuvi galvenajai sistēmai. Šajā gadījumā tiek regulāri mērīta sistēmas funkcionalitātes ātrdarbība, automātiski regulējot, cik daudz papildu lietotājiem atkļaut piekļuvi konkrētajam sistēmas modulim. Būtībā līdzīgos apstākļos abās situācijās paralēlajam atļauto lietotāju skaitam būtu jābūt līdzīgam, tomēr realitātē situācija mēdz būt citādāka. Situāciju mēdz ietekmēt dažādi apstākļi – kaut vai lietotāju konkrētā laika sistēmas lietošanas paradumi. Var būt situācija, kad kādas konkrētas resursu ietilpīgas atskaites tiek ģenerētas mēneša pirmajās dienās. Tādā gadījumā pat ja konkrētais rindu mehānisma algoritms regulē nodaļu un lietotāju profilu datu apstrādi, kas nav saistīta ar resursus prasošās atskaites ģenerēšanu, paralēli esošā atskaišu ģenerēšana patērē lielu daļu sistēmas resursu, palēninot visu pārējo moduļu darbību. Rezultātā šo atskaišu ģenerēšanas laikā pieņemamas ātrdarbības saglabāšanai citos moduļos būs mazāks paralēli atļauto lietotāju skaits salīdzinot ar laiku, kad resursu ietilpīgās atskaites netiek ģenerētas.

Lai noteiktu, cik daudziem lietotājiem pilnībā automatizētās piekļuves gadījumā atļaut pievienoties jau esošajiem lietotājiem, tiek izmantota salīdzināšana starp vidējo patērēto laiku konkrētā moduļa ielādei un atļauto maksimālo ielādes laiku. Piemēram, ja vidējais ielādes laiks ir sasniedzis astoņas sekundes pie x aktīvajiem lietotājiem pēdējo desmit minūšu laika griezumā un rindas slieksnis ir desmit sekundes, tad pašreizējo lietotāju skaitu algoritms palielina par

1-8/10=>100%-80%=>20%.

Tas nozīmētu, ka tūkstoš paralēlu lietotāju gadījumā pie nākamās rindā esošo lietotāju ievietošanas atļauto lietotāju sarakstā uz minēto sarakstu tiks pārvietoti papildu divi simti lietotāji. Kaut arī sākotnēji izskatās, ka šādi paralēlo lietotāju skaits nemitīgi pieaugs, tomēr tā

nav patiesība, jo paralēli katru minūti no atļauto lietotāju saraksta tiek izņemti tie lietotāji, kas pēdējo divdesmit minūšu laikā moduli nav izmantojuši. Tādējādi tiek panākts pašregulējošs mehānisms, kurš automātiski pielāgojas gan esošajai servera noslodzei, gan nodrošina efektīvu rindas pārvaldību.

4.15. Mērījumos izmantotās sistēmas specifika

Lai pārliecinātos par izstrādātā risinājuma efektivitāti un lietojamību reālas sistēmas apstākļos, rindu regulējošais mehānisms tika integrēts un testēts 2People.com informāciju sistēmas izstrādes vidē. 2P ir mākoņservisa bāzēta, programmatūra kā pakalpojums uzņēmumu iekšējā HR sistēma, kas nodrošina gan uzņēmuma visu HR procesu atbalstu, gan dažādu veidu iekšējo uzskaiti, gan sarežģītu lietotāju tiesību mehānismu, gan resursietilpīgu atskaišu un biznesa pārraudzības moduli. 2P gadījumā viena sistēma vienā infrastruktūrā apkalpo daudzas kompānijas – tādējādi katra jauna bizness biznesam (*business to business,* B2B) klienta piesaistīšana palielina 2P lietotāju skaitu no dažiem desmitiem līdz pat vairākiem tūkstošiem.

Maģistra darba autores ieguldījums minētās sistēmas izstrādē ir iesaistīšanās sistēmas testēšanas plānošanā un realizēšanā. 2P kā testa sistēma tika izvēlēta tāpēc, ka tās tuvāko divu gadu laikā paredzētais lietotāju skaits plānots desmitos tūkstošu, pie tam, sistēmas pašreizējā specifika ir tāda, ka dažas reizes gadā sistēmai ir ļoti izteiktas lietošanas pīķa stundas, kamēr pārējā laikā sistēmas aktīvo lietotāju skaits ir neliels. Otrs iemesls un specifika ir sistēmā izstrādes procesā esošais integrāciju atskaišu modulis, kuram tiešsaistē jāizanalizē liels datu apjoms, kas pašreizējā struktūrā ir laika un resursu ietilpīgs process. Līdz ar to 2P sistēmas gadījumā bija nepieciešamība pārliecināties, vai aprakstītā rindu mehānisma risinājums varētu nākotnē pieaugoša sistēmas lietotāju skaita gadījumā nodrošināt sistēmas lietojamību pieņemamā līmenī.

4.16. Izstrādes darbu apraksts

Pats izstrādes darbs tika veikts komandā, sadarbojoties darba autorei ar 2P sistēmas izstrādātājiem. Maģistra darba ietvaros tika izstrādāts rindu mehānismu regulējošās sistēmas prototips, kas tika integrēts 2P izstrādes vides sistēmā, pielietojot risinājumu konkrētam atskaišu moduļa blokam. Darba autore veica rindu mehānisma teorētiskā un praktiskā modeļa, atbilstošās

datu bāzes un saistīto servisu izstrādi, kamēr integrāciju 2P sistēmā veica 2P sistēmas izstrādātāji. Pēc integrācijas pabeigšanas kopīgi tika veikta sistēmas noslodzes modelēšana un noslodzes testi, lai pārliecinātos, ka minētais risinājums spētu novērst liela lietotāju skaita gadījumā radušos sistēmas ātrdarbības traucējumus. Pēc maģistra darba mērījumu pabeigšanas mērījumi tika iesniegti sistēmas izstrādes vadītājam un īpašniekiem, lai izvērtētu potenciālos ieguvumus un lemtu par risinājuma ieviešanas nepieciešamību produkcijas vidē.

4.17. Slodzes testēšana

Sākotnēji risinājums tika uzstādīts sistēmas testa vidē, lai ievāktu reālus izpildes laika datus no vairākiem sistēmas moduļiem. Ievāktie dati tika izmantoti rindu mehānisma izstrādē, lai:

- Izprastu atšķirības starp dažādiem moduļiem;
- Izstrādātu žurnālfailu ielasīšanu;
- Sagatavotu rindu mehānisma API.

Pēc testa vides datu izanalizēšanas tika secināts, ka pie pašreizējā lietotāju skaita rindu mehānisma efektivitāti pārbaudīt produkcijas vidē tuvākajā laikā nebūs iespējams. Vienlaikus bija skaidrs, ka nākotnē sistēmai attīstoties problēma kļūs aizvien aktuālāka. Tika izlemts par nepieciešamību eksperimentāli radīt dažādas sistēmas slodzes, lai saprastu, kurā mirklī rindu mehānisms būs noderīgs un vai tā ieviešana atrisinās gaidāmas lietojamības problēmas.

Lai notestētu rindu mehānismu darbībā tika izlemts, ka nepieciešams veidot lielu skaitu paralēlu pieprasījumu no sistēmas lietotājiem. Turpmākie galvenie risināmie jautājumi bija:

- Izpētīt un atrast, kuru rīku izmantot noslodzes testēšanai;
- Sagatavot lietotāju sarakstu;
- Sagatavot lietotāju autentifikācijas marķierus testu veikšanai.

Lai nebūtu jāgatavo lietotāju autentifikācijas marķierus katram no testa lietotājiem, tika veiktas izmaiņas sistēmas izstrādes vides iestatījumos, kas ļāva ar autentificētu lietotāja marķieri piekļūt sistēmas funkcionalitātei gan savā, gan citu sistēmas lietotāju vārdā. Pirms funkcionalitātes izsaukuma, izmantojot gadījumskaitļu ģenerēšanas funkciju, tika izvēlēts kāds no iepriekš definētajiem lietotājiem, kurš konkrētajā brīdī aizstāja autentifikācijas marķiera norādīto lietotāju. Tādējādi tika nosimulēta reālas sistēmas slodze, izmantojot tikai vienu autentifikācijas marķiera vērtību.

Attiecībā uz rīku izvēli tika apskatīti trīs dažādi Google meklētājā atrasti vai ikdienā jau lietoti testēšanai paredzēti rīki – Locust, Postman un Loadtest. No funkcionalitātes viedokļa ļoti daudzsološu iespaidu atstāja Locust, tomēr Locust gadījumā nepieciešamā testēšanas rīka konfigurēšana aizņem vairāk laika salīdzinot ar abiem pārējiem apskatītajiem. Līdz ar to, kaut arī Locust ir funkcijām bagātāks risinājums, tomēr rindu mehānisma žurnālfailu iegūšanas mērķa sasniegšanai pilnībā pietiekoši bija ar abiem pārējiem – vienkāršākajiem rīkiem.

Kā pirmais testu veikšanai tika izmantots Postman. Kaut arī Postman ļāva nosūtīt lielu skaitu pieprasījumu konkrētai funkcionalitātei, tomēr pēc testu rezultātu apskatīšanas radās aizdomas, ka Postman veiktie pieprasījumi neizpildās īsti paralēli. Ar laiku radušās šaubas apstiprinājās – Postman katru nākamo pieprasījumu izsūtīja secīgi pēc iepriekšējā pieprasījuma atbildes saņemšanas. Tādējādi Postman nevarēja tikt izmantots lielas paralēlās slodzes testu veikšanai.

Visbeidzot tika izmēģināts komandrindas bāzētais sistēmu testēšanas rīks Loadtest (*https://www.npmjs.com/package/loadtest*). Loadtest atšķirībā no abiem pārējiem izskatījās pēc mazāk attīstīta rīka, tomēr tā iespēju izpēte liecināja, ka rīks ir piemērotākais no apskatītajiem rīkiem minētā testa veikšanai. Loadtest gadījumā ir iespējams norādīt daudzus ievadparametrus, kas šī testa ietvaros ļāva norādīt, cik daudz paralēlo pieprasījumu nepieciešams veikt un kādu skaitu pieprasījumu kopumā nepieciešams veikt. Pēc žurnālu faila rezultātiem bija skaidrs, ka pieprasījumi tiek veikti gan paralēli, gan negaidot iepriekšējo pieprasījumu atbildes, tāpat bija redzams, ka žurnālu failā uzkrājas turpmākai analīzei nepieciešamā informācija.

4.18. Prototipa aprobācija

Izveidotā rindu kontroles mehānisma pārbaude tika veikta ar slodzes testa palīdzību. Testa galvenais mērķis bija novērtēt, vai izveidotais rindu mehānisms ļauj vadīt lietotāju radīto slodzi sistēmai. Lai novērtētu izveidotā rindu mehānisma darbības efektivitāti, slodzes testi tika veikti gan ar ieslēgtu rindu mehānismu, gan arī ar izslēgtu. Testā ar izslēgtu rindu mehānismu tika izmantoti 34 lietotāju ID (100% no lietotājiem ir atļauta piekļuve sistēmas modulim). Savukārt, testā ar ieslēgtu rindu mehānismu 2 lietotājiem tika atļauta piekļuve izvēlētajam modulim, savukārt pārējie tika ievietoti gaidītāju rindā (2/34=6%, tādējādi 6% no lietotājiem ir atļauta piekļuve izvēlētajam sistēmas modulim). Jāņem arī vērā, ka reālā lietošanas situācijā rindā ievietotie lietotāji neveiktu nepārtrauktu lapas pārlādi, līdz ar to reālā dzīvē rindu mehānisma

ieguvums varētu būt vēl lielāks, jo katrs unikālais lietotājs, nonākot rindā, veic krietni mazāk pieprasījumus sistēmā.

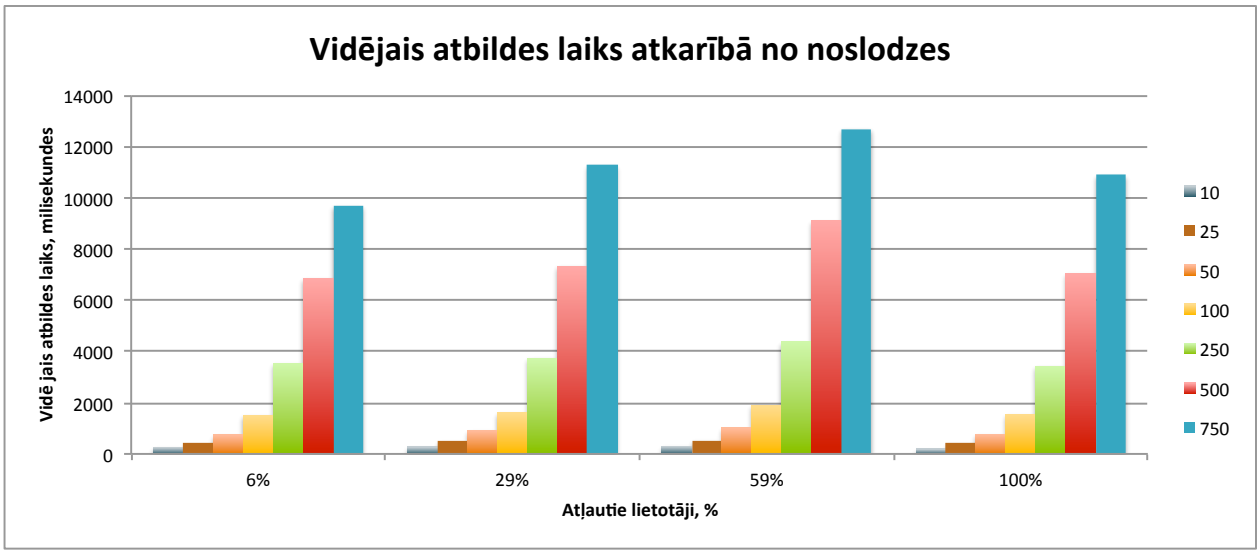
Veicot slodzes testu, žurnālfailos tika reģistrēts pieprasījuma atbildes statuss, kas attiecīgi bija vai nu *completed* (funkcionalitātes moduļa izsaukums ir saņemts un laikietilpīgais programmatūras kods ir izpildīts uz servera) vai *waiting* (funkcionalitātes moduļa izsaukums uz servera ir saņemts, bet lietotājs ir ievietots gaidītāju rindā bez izsauktās laikietilpīgās darbības izpildes). Abas atbildes testa analīzes rezultātos tiek uzskatītas par apmierinošām, jo norāda, ka abos gadījumos serveris sniedz atbildi lietotājam un pārslodzes dēļ nenotiek sistēmas atteice. Rindu mehānisma pilnā darbības plūsmā reālos lietošanas apstākļos, tikko servera slodze ir mazinājusies zem noteiktas sliekšņa vērtības, daļa no lietotājiem, kas atrodas rindā, tiek ievietota atļauto lietotāju sarakstā un var piekļūt izvēlētajam sistēmas modulim.

Tika izmantota rīka sniegtā iespēja sūtīt pieprasījumus pakotnēs (*-c, concurrency*), pie tam katra nākamā pakotne tika nosūtīta negaidot, kamēr ir saņemta atbilde par iepriekšējā pieprasījuma apstrādi. Testā gan ar ieslēgtu, gan izslēgtu rindu mehānismu tika izmantotas vērtības 10, 25, 100, 250, 500, 750 vienlaicīgi pieprasījumi pakotnē.

Katrā testā tika uzstādīts noteikts izsūtāmo pieprasījumu skaits – 2000 (*-n,* detalizētu izsaukuma un atbildes piemēru skat. 3. pielikumā). Tomēr, tā kā tika izmantota iespēja sūtīt pieprasījumus noteiktās pakotnēs, tad, saskaņā ar aprakstīto rīka specifikāciju, reālais nosūtītais pakotņu skaits ir lielāks, kas arī tika novērots praktiski izpildot testu. Kopējais analizējamo pieprasījumu skaits – 2000 katram pakotņu un testa (ar un bez QCM) veidam – ir pietiekošs, lai saprastu tendenci, kas nepieciešama rindu mehānisma efektivitātes novērtēšanai.

Slodzes tests tika veikts laikā, kad servera noslodze bija maksimāli zema, lai pēc iespējas vairāk norobežotos no citiem faktoriem, kas varētu ietekmēt rezultātus. Slodzes testu rezultāti tika ierakstīti QCM datubāzē un izmantoti turpmākai iegūto rezultātu analīzei.

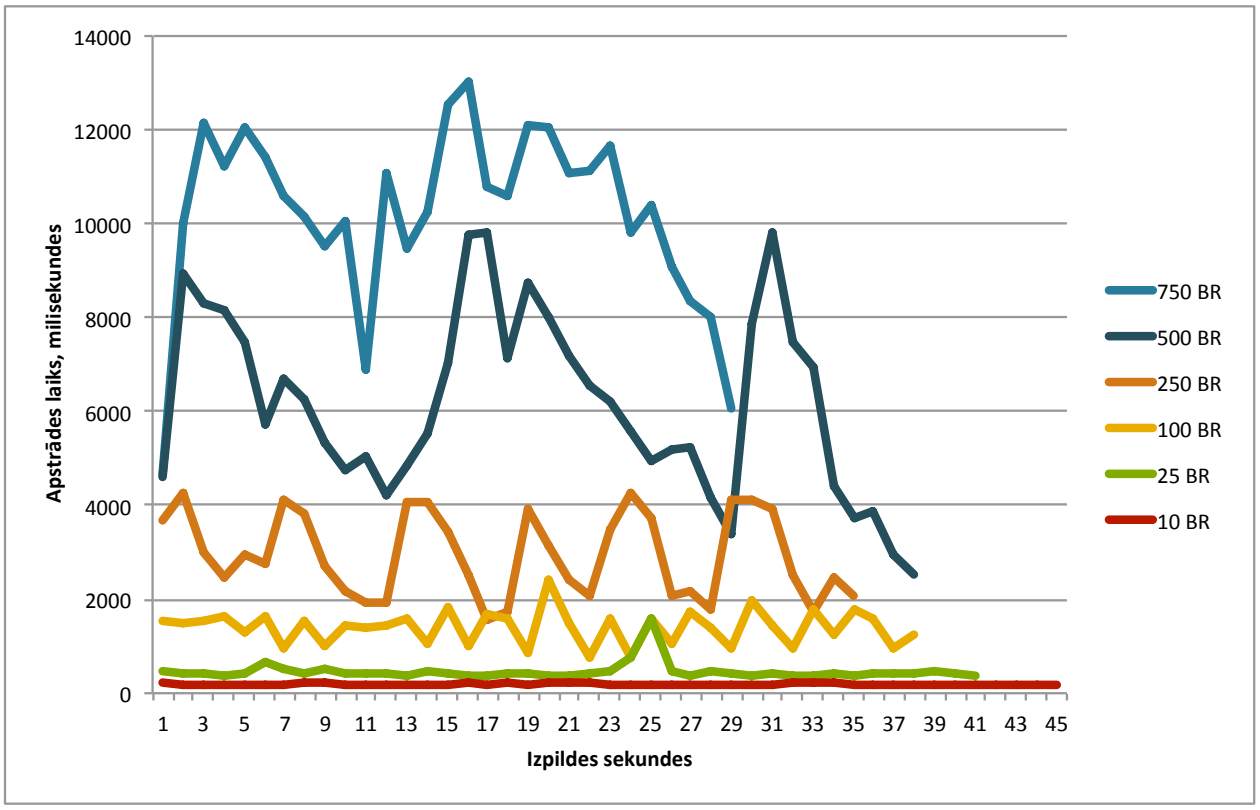
Sākotnēji tika apskatīts kopējais pieprasījumu izpildes laiks (gan pieprasījumiem ar statusu *completed,* gan ar statusu *waiting*) atkarībā no sistēmas noslodzes (testā ģenerēto pieprasījumu skaita pakotnē). Attēlā 4.7 ir redzama tendence, ka neatkarīgi no atļauto lietotāju skaita, jo lielāks paralēlo pieprasījumu skaits pakotnē un līdz ar to augstāka servera noslodze, jo ilgāks laiks nepieciešams pieprasījuma apstrādei neatkarīgi no tā, vai rindu mehānisms darbojas vai nē (100% atļautie lietotāji).



4.7. att. **Vidējais atbildes laiks atkarībā no paralēlo pieprasījumu skaita**

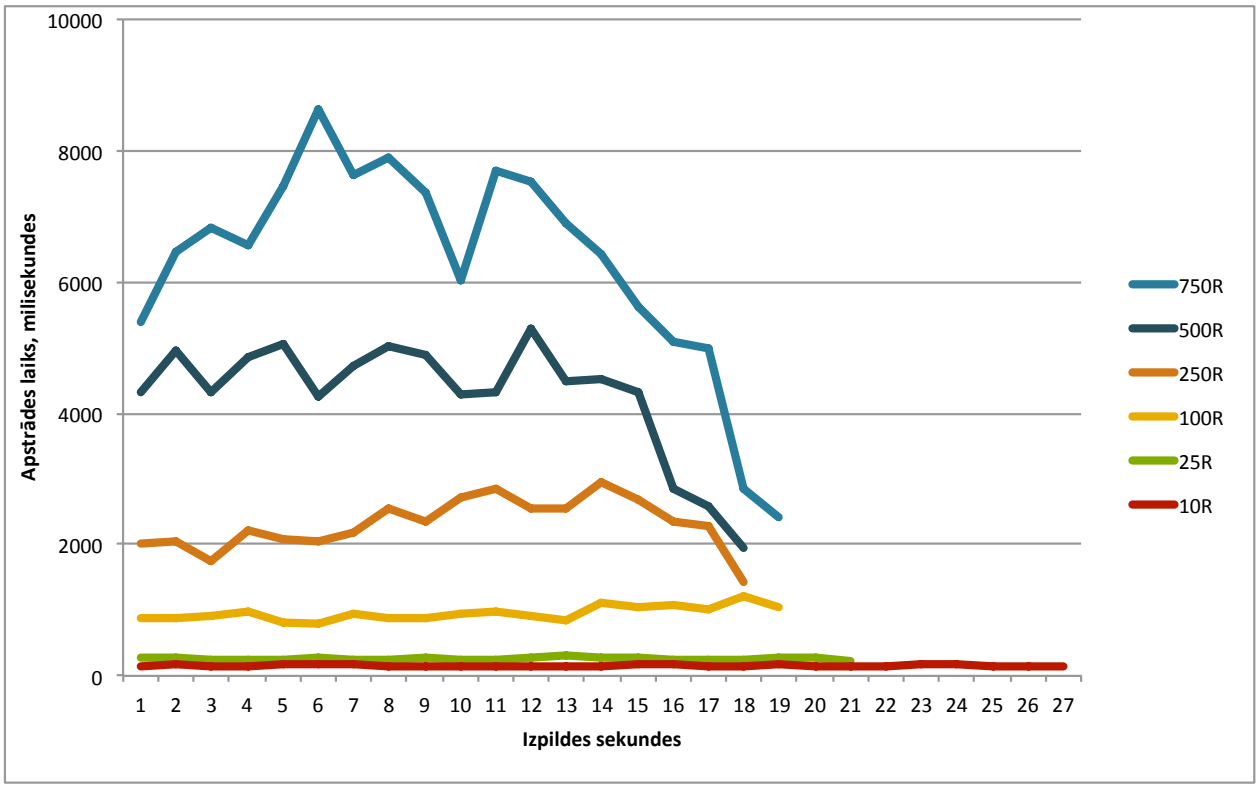
Lai novērtētu precīzāku rindu mehānisma ietekmi uz aizsargājamo sistēmu, tika apskatīts vidējais izpildes laiks katram slodzes apjomam atsevišķi. Pie tam, tas ir darīts tikai pieprasījumiem ar statusu *completed* tādējādi mēģinot novērtēt, kā rindu mehānisma ieviešana ietekmē pieprasījumu apstrādes ātrumu laikā, kad pārējie lietotāji ir ievietoti gaidītāju rindā un nevar piekļūt noslogotajam modulim. Pieprasījumu skaits ar statusu *completed* katrā no rindu mehānisma testiem atšķiras, tās ir attiecīgi 6%, 29%, 59% un 100% no kopējām testā izmantotā pieprasījumu skaita (detalizēts pieprasījumu skaita sadalījums ir apskatāms 4.pielikumā).

Attēlā 4.8 tiek apskatīts vidējais apstrādes laiks katrā no izpildes sekundēm (1., 2., 3., …) sistēmas modulim bez rindu mehānisma ar dažādu paralēlo pieprasījumu skaitu katrā pakotnē. Apskatītajā gadījumā piekļuve izvēlētajam sistēmas modulim ir visiem lietotājiem. Kopējā tendence – jo lielāka pakotne, jo ilgāk ir jāgaida atbilde no servera – šeit saglabājas. Piemēram, 13. testa izpildes sekundē, vidējais apstrādes laiks ar 100 paralēliem pieprasījumiem pakotnē bija apmēram 1.6 sekundes (1594 milisekundes), ar 250 paralēliem pieprasījumiem pakotnē – apm. 4 sekundes (4077 milisekundes), ar 500 paralēliem pieprasījumiem pakotnē – apm. 4.9 sekundes (4873 milisekundes) un savukārt, ar 750 – apm. 9.4 sekundes (9455 milisekundes).



4.8. att. **Vidējais pieprasījuma apstrādes laiks bez ieslēgta rindu mehānisma**

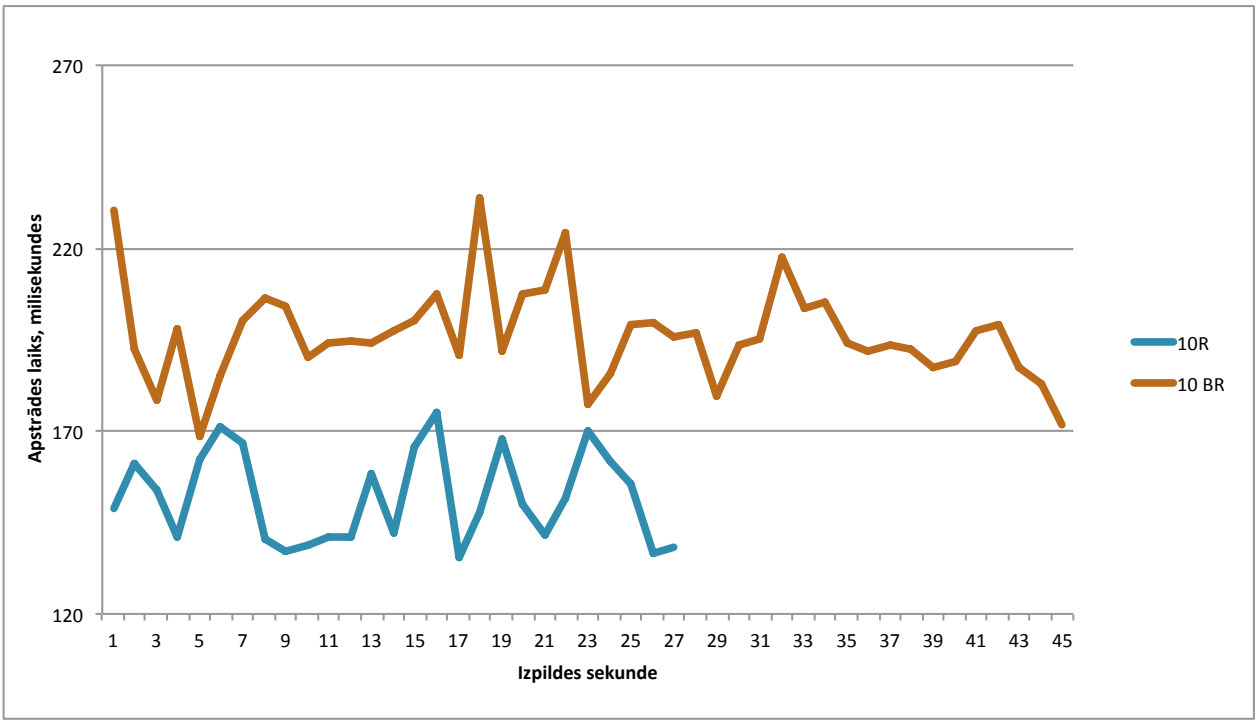
Nākamajā testā ir aktīvs rindu veidošanas mehānisms un piekļuve sistēmas modulim tiek dota tikai 6% lietotāju. Apskatot vidējo apstrādes laiku katrā no izpildes sekundēm (1., 2., 3., …) sistēmas modulim ar ieslēgtu rindu mehānismu dažādiem paralēlo pieprasījumu skaitiem (skat. att. 4.9), ir redzama tāda pati tendence kā iepriekšējos gadījumos – jo lielāka pakotne (vienlaicīgi iesūtīto pieprasījumu skaits), jo ilgāks apstrādes laiks ir nepieciešams. Lielākās pakotnes (750 paralēli pieprasījumi vienā pakotnē) gadījumā 6% lietotāju atbilst 148 pieprasījumi ar statusu *completed* (skat. 4. pielikumu). Šajā gadījumā 13. testa izpildes sekundē, vidējais apstrādes laiks ar 100 paralēliem pieprasījumiem pakotnē bija apmēram 0.8 sekundes (840 milisekundes), ar 250 paralēliem pieprasījumiem pakotnē – apm. 2.5 sekundes (2554 milisekundes), ar 500 paralēliem pieprasījumiem pakotnē – apm. 4.5 sekundes (4493 milisekundes) un savukārt, ar 750 – apm. 6.9 sekundes (6892 milisekundes). Šeit ir novērojama tendence, ka pie lielākas slodzes, tā daļa pieprasījumu, kas nav ievietota gaidītāju rindā, tiek ātrāk apstrādāta nekā gadījumā bez rindu mehānisma.



4.9 att. **Vidējais pieprasījuma apstrādes laiks ar rindu mehānismu**

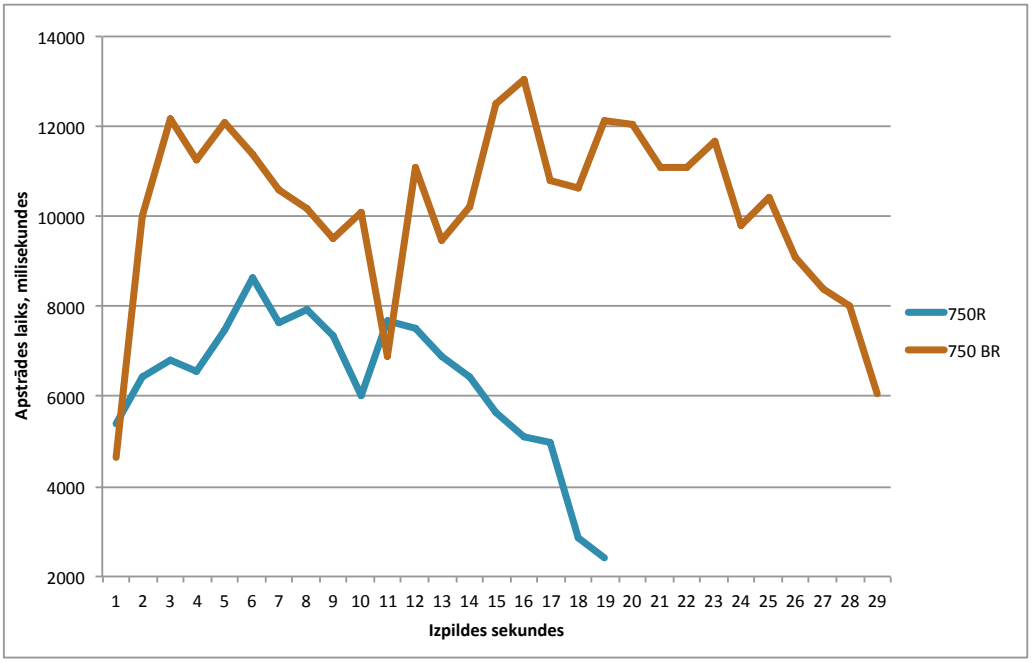
Testa apstākļos gan ar, gan bez rindu mehānisma, pakotnēm ar mazāku paralēlo pieprasījumu skaitu ir ilgāks kopējais testa izpildes laiks. Kopējais testa izpildes laiks pakotnei ar 10 paralēliem pieprasījumiem bija visilgākais – bez rindu mehānisma tas aizņēma 45 sekundes, ar – 27 sekundes (att. 4.10). Atšķirība kopējā testa izpildes laikā visticamāk ir skaidrojama ar to, ka rindu mehānisma gadījumā sarežģīta aizsargājamās sistēmas funkcionalitāte ir jāizpilda tikai 6% no kopējā pierasījumu skaita pretstatus 100% gadījumam bez rindas.

Apskatot sistēmas atbildes laiku milisekundēs testa datiem, kur nosūtīto paralēlo pieprasījumu skaits pakotnē bija neliels – 10 pieprasījumi – ir redzams, ka apstrādes laiks vidēji atšķiras apmēram par 50 milisekundēm (skat. att. 4.10, sarkanā līnija attēlo vidējo apstrādes laiku katrā izpildes sekundē bez rindu mehānisma (kopā 2009 pieprasījumi), zilā līnija – ar ieslēgtu rindu mehānismu, kur piekļuve atļauta 6 % (105) pieprasījumu). Tādējādi pie nelielas noslodzes ieguvums no rindu mehānisma izmantošanas sistēmā ir neliels un pie noteikta pieprasījumu apjoma, serveris spēj apkalpot arī pieprasījumus bez rindas. Tomēr pieaugot slodzei, apstrādes laiks palielinās un rindu mehānisms kļūst efektīvs – sistēma tiek pasargāta no atteicēm, kā arī lietotāji var veikt izvēlētās darbības vai nu uzreiz vai pēc gaidīšanas rindā.



4.10 att. **Vidējais pieprasījuma apstrādes laiks pie nelielas servera noslodzes**

Savukārt, palielinoties pieprasījumu skaitam pakotnē, sistēma ar rindu mehānismu spēj efektīvāk apstrādāt iesūtītos pieprasījumus. Attēlā 4.11 ir atainots vidējais apstrādes ilgums katrā izpildes sekundē testa mērījumiem ar 750 pieprasījumiem pakotnē pie atļautiem 6% (148) lietotāju (2286 pieprasījumi saņem atbildi no servera par atrašanos *waiting* statusā).

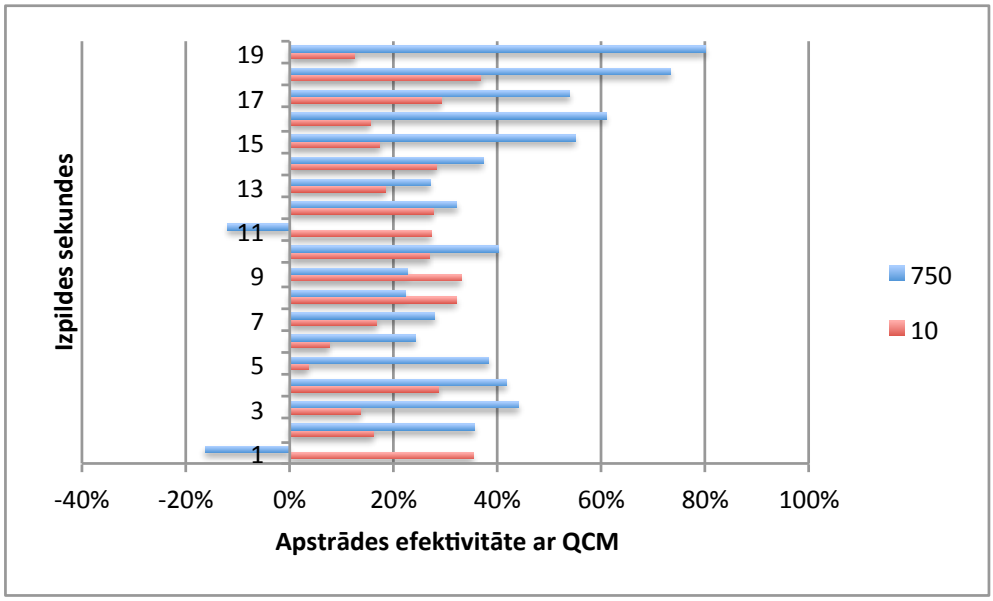


4.11 att. **Vidējais pieprasījuma apstrādes laiks pie liela pieprasījumu skaita**

Pieprasījumu apstrāde 4.11 attēlā aplūkotajā gadījumā ir notikusi ātrāk, kas reālā situācijā ļautu piekļuvi jauniem lietotājiem no gaidītāju rindas.

Apskatot iegūtos testu datus, ir novērojama tendence, ka ar esošajiem parametriem sistēma ar aktīvu rindu mehānismu spēj apstrādāt konkrētos pieprasījumus par vidēji 30% līdz 50% ātrāk (pieprasījumiem, kas ir ar statusu *completed*). Attēlā 4.12 ir attēlota pieprasījumu efektivitāte ar ieslēgtu rindu mehānismu (6% lietotāju atļauta piekļuve) testa datiem ar 10 un 750 pieprasījumiem pakotnē. No iegūtajiem datiem ir secināms, ka lielākas slodzes gadījumā rindu mehānisma efektivitāte ir lielāka (zilie stabiņi grafikā) nekā pie mazākas slodzes (sarkanie stabiņi grafikā). Piemēram, apskatot attēlu 4.12 un izvērtējot testa izpildes 16. sekundi, ir novērojams, ka lielāka paralēlo pieprasījumu skaita pakotnē (750) gadījumā ar rindu mehānismu pieprasījumu apstrāde ir par aptuveni 60% ātrāka nekā bez rindu mehānisma (šis rādītājs tika aprēķināts kā efektivitāte = 1 - <vidējais apstrādes laiks izpildes sekundē ar rindu mehānismu>/ <vidējais apstrādes laiks izpildes sekundē bez rindu mehānisma>). Šajā pašā izpildes sekundē pieprasījumu apstrāde testam ar 10 paralēliem pieprasījumiem pakotnē ar rindu mehānismu ir par 15% ātrāka nekā bez ieslēgta rindu mehānisma.

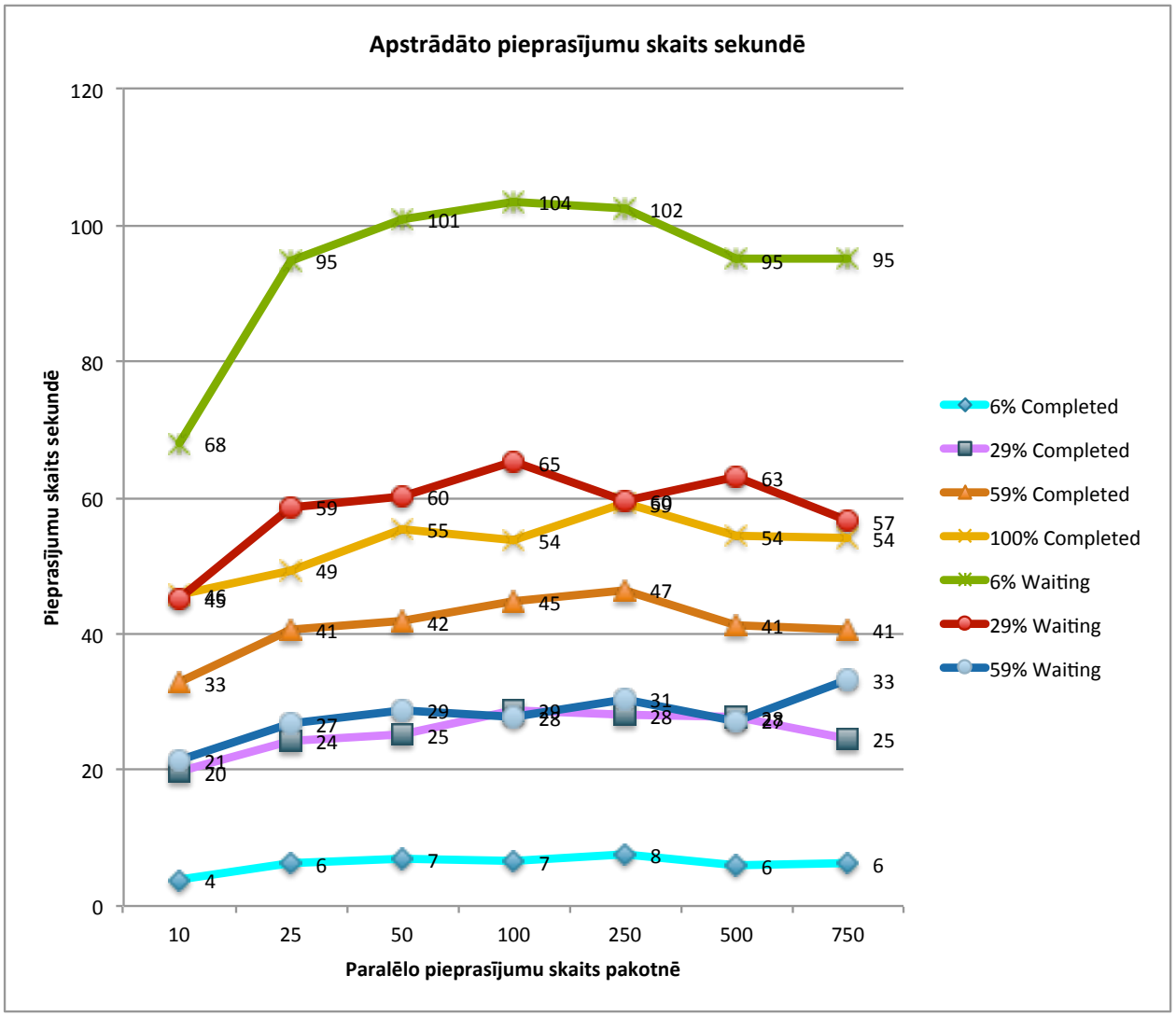
Divos izpildes laikos (1. un 11. izpildes sekundē) efektivitāte ar ieslēgtu rindu mehānismu tomēr ir negatīva – iespējamie šādu rezultātu iemesli ir lielais pieprasījumu ielādes skaits testa sākumā (noris pirmā ielādes sekunde) un traucējumiem savienojumā ar serveri vai pašā servera darbībā. Tā kā šāda situācija neatkārtojās, sīkāka situācijas analīze netika turpināta.



4.12 att. **Vidējais pieprasījuma apstrādes laiks pie liela pieprasījumu skaita**

Apskatītie rezultāti parāda rindu mehānisma darbību divās robežsituācijās – kad piekļuve ir sniegta 100% lietotāju (jeb rindu mehānisms ir izslēgts) un 6% lietotāju (sistēmas piekļuve ir atļauta 2 lietotājiem no 34).

Analizējot testā iegūtos rezultātus, tika apskatīts arī apstrādāto pieprasījumu skaits sekundē (skat. att. 4.13). Grafikā pieprasījumi ar statusu *completed* un *waiting* ir attēloti atsevišķi, pie tam katram ir norādīts arī atļauto lietotāju procentuālais apjoms.



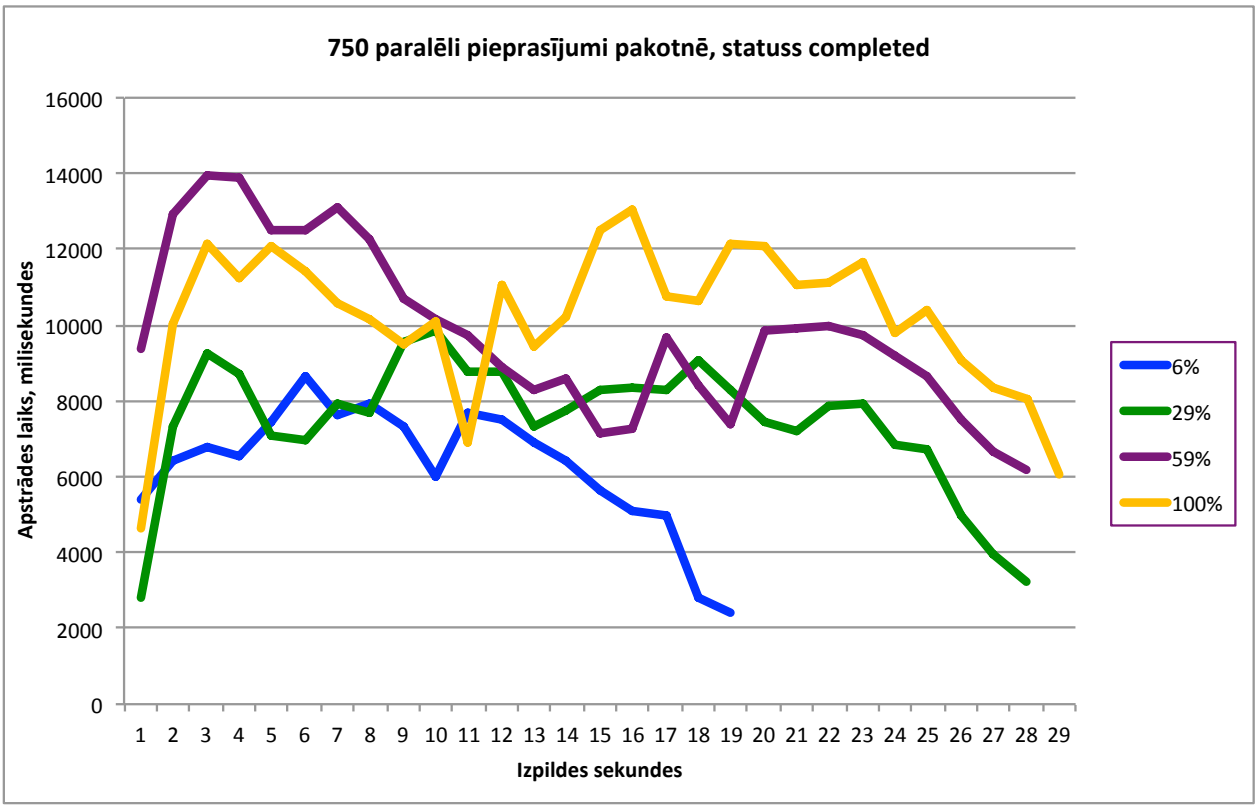
4.13 att. **Pieprasījumu skaits sekundē atkarībā no pakotnes izmēra un atļauto lietotāju skaita**

Lielākais apstrādāto pieprasījumu skaits sekundē ir pie 6% un 29% atļauto lietotāju un statusa *waiting* (serveris pieprasījumu ir saņēmis, lietotājs ievietots gaidītāju rindā), savukārt, vismazākais apstrādāto pieprasījumu skaits sekundē ir pie 6% atļauto lietotāju un statusu *completed* (skat. att. 4.13). Ir novērojama tendence, ka apstrādāto pieprasījumu skaits sekundē pieprasījumiem ar statusu *completed* ir zemāks nekā pieprasījumiem ar statusu *waiting*. Tā

galvenais iemesls ir atšķirība izpildāmajā funkcionalitātē. Pārbaude un ievietošana rindā gandrīz vienmēr būs krietni ātrāks process par sarežģītas un laikietilpīgas funkcionalitātes izpildi (piemēram, atskaišu ģenerēšana).

Apskatot attēlu 4.13, nav novērojamas izteikti krasas atšķirības apstrādāto pieprasījumu skaitam sekundē atkarībā no paralēlo pieprasījumu skaita pakotnē, izņemot 6% *waiting* gadījumu, kurā pakotnei ar 10 paralēliem pieprasījumiem ir izteikti zemāks apstrādāto pieprasījumu skaits sekundē salīdzinot ar citiem pakotņu izmēriem pie 6% atļautiem lietotājiem. Tas, visticamāk, skaidrojams ar servera spēju efektīvi apstrādāt arī vairāk kā 10 paralēlas pieprasījumu pakotnes neatkarīgi no rindu mehānisma.

Testi tika veikti arī apskatot sistēmas darbību, kad piekļuve ir atļauta 29% (10 lietotājiem atļauta piekļuve sistēmai, pārējie 24 rindā) un 59% (20 lietotājiem atļauta piekļuve izsaucamajam sistēmas modulim, pārējie 14 rindā) lietotāju (attēls 4.14). Attēlā ir apskatīts vidējais pieprasījumu izpildes laiks katrā testa sekundē ar statusu *completed*. Rezultātu analīzei tika izmantoti testa dati, kuros pakotnē bija 750 paralēli pieprasījumi.



4.14 att. **Pieprasījumu ar statusu *completed* apstrādes laiks atkarībā no atļauto lietotāju apjoma**

Attēlā 4.14 ir redzama tendence, ka pieprasījumu vidējais apstrādes laiks atļautiem 29% un 59% tiecas atrasties starp rezultātu līknēm, ko veido pieprasījumu apstrādes laiku dati 6% atļauto lietotāju (attēlā zilā līnija, viens no īsākajiem pieprasījumu apstrādes laikiem) un 100% atļauto lietotāju (attēlā dzeltenā līnija, viens no garākajiem pieprasījumu apstrādes laikiem). Šeit ir redzama tendence – jo mazāks atļauto lietotāju skaits un līdz ar to mazāks pieprasījumu skaits un mazāka servera noslodze, jo ātrāk tiek apstrādāti pieprasījumi (attēlā zilā līnija, vidējais pieprasījumu apstrādes laiks nepārsniedz 8.6 sekundes). Jo ātrāk tiek apstrādāti pieprasījumi, jo lielāka iespēja ir dot piekļuvi aizsargājamās sistēmas funkcionalitātei nākamajiem rindā gaidošajiem lietotājiem.

REZULTĀTI

Maģistra darba rezultātā ir aprakstīts, izveidots un aprobēts prototips pašregulējošai aizsardzībai pret sistēmas pārslodzi – ir izstrādāts no sistēmas veiktspējas atkarīgs rindu veidošanas mehānisms. Izveidotais mehānisms ļauj samazināt lietotāju radīto slodzi sistēmai un kontrolēti ļaut piekļuvi noslogotākajiem sistēmas funkcionalitātes moduļiem, minimāli ietekmējot pārējo sistēmas darbību. Risinājumā tiek veidotas gaidītāju rindas jaunajiem lietotājiem, tādējādi ļaujot efektīvi darboties jau sistēmā esošajiem lietotājiem. Lietotāji, kas vēlas piekļūt sistēmas modulim ar izveidojušos rindu, tiek informēti par aptuveno gaidīšanas laiku, pēc kura būs iespējams piekļūt izvēlētajai sistēmas funkcionalitātei. Līdzko slodze uz servera ir mazinājusies (pieprasījuma izpildes laiks ir zem noteiktās sliekšņa vērtības), tiek dota piekļuve nākamajiem gaidītājiem no izveidotās rindas.

Izstrādātais rindu mehānisms ne tikai pasargā sistēmu no pārslodzes, ko var radīt pārāk liels pieprasījumu skaits, bet arī samazina katra ienākošā pieprasījuma apstrādes laiku salīdzinot ar situāciju, kad sistēma darbojās bez aktīva rindu mehānisma. Iegūtajos testu datos ir novērojama tendence, ka rindu mehānisma efektivitāte palielinās pieaugot lietotāju pieprasījumu skaitam (servera noslodzei).

Pēc maģistra darbā izstrādātā rindu veidošanas mehānisma darbības analīzes var secināt, ka izmantojot izstrādāto produktu, ir iespējams nodrošināt pašregulējošu aizsardzību pret sistēmas pārslodzi.

SECINĀJUMI

Maģistra darba ietvaros apskatītā problēma ir aktuāla ne tikai apskatītajā personāla vadības sistēmā, bet arī citās, plašam lietotāju skaitam pieejamās, sistēmās, piemēram, Dziesmu un Deju svētku biļešu tirdzniecības, eVeselības un Valsts ieņēmumu dienesta EDS sistēmās.

Izstrādātā pašregulējošā aizsardzības mehānisma pret sistēmas pārslodzi risinājuma viens no plusiem ir salīdzinoši zemās izmaksas tā ieviešanai, jo nav nepieciešams pārbūvēt visu regulējamo sistēmu. Tāpat modulis ir samērā vienkārši pielāgojams izmantošanai citās sistēmās. Protams, rindu veidošanas mehānisms ir tikai viens no potenciālajiem problēmas risināšanas veidiem. Jāņem vērā, ka lietotāja pieprasījumu likšana rindā ir sava veida lietotāja ierobežošana, jo ir nepieciešams pagaidīt, līdz varēs piekļūt izvēlētajai funkcionalitātei. Risinājums iespējams, nav arī pats populārākais, jo var tikt izmantotas citas iespējas, piemēram, tehniskā nodrošinājuma uzlabošana vai funkcionalitātes optimizācija. Tomēr maģistra darba ietvaros izveidotais rindu veidošanas mehānisms ir labs un ātrs palīgs situācijās, kad ierobežota laika un budžeta apstākļos ir jāsagatavojas noteiktam lietotāju skaitam, kas izmantos konkrētu sistēmas funkcionalitāti.

Izveidotais pašregulējošais aizsardzības mehānisms pret sistēmas pārslodzi ir pielietojams atsevišķiem sistēmas funkcionalitātes moduļiem, ne tikai visai sistēmai kopīgi. Tas dod iespēju rindu mehānismu ieviest tikai visvairāk resursus prasošajiem funkcionalitātes moduļiem ar paredzamu lielu lietotāju skaitu, pārējiem sistēmas moduļiem dodot brīvu piekļuvi.

Atkarībā no situācijas, izveidoto rindu veidošanas mehānismu ir iespējams darbināt gan uz tā paša servera, gan uz cita, tādējādi to pilnībā atdalot no ekspluatējamās vides un līdz ar to minimāli ietekmējot galvenās sistēmas efektivitāti.

Darba mērķis izstrādāt prototipu un izvērtēt tā efektivitāti ir sasniegts. Iegūtie rezultāti parāda, ka izveidotais rindu mehānisms ļauj samazināt slodzi uz sistēmu un nodrošina kontrolētu lietotāju piekļuvi izvēlētiem sistēmas moduļiem. Lai noteiktu precīzāk, kad nepieciešams dot piekļuvi un cik daudziem lietotājiem, būtu nepieciešami reāli lietošanas dati par ilgāku laika posmu.

Pētījuma rezultāti ir iesniegti arī 2P sistēmas pasūtītājiem, lai izvērtētu, vai esošajam lietotāju skaitam ir nepieciešams ieviest sistēmu arī produkcijā un, ja jā, tad izvērtēt, kuriem sistēmas funkcionalitātes moduļiem tas būtu nepieciešams.

PATEICĪBAS

Vēlos izteikt pateicību savam vīram Valdim Takerim par uzmundrinājumu, atbalstu un vērtīgiem komentāriem visā maģistra darba izstrādes laikā. Vēlos teikt paldies arī pārējai manai ģimenei, kas pieņēma un atbalstīja manu izvēli, un izturējās ar sapratni un iecietību pret to, ka darba rakstīšana prasa prombūtni un laiku.

IZMANTOTĀ LITERATŪRA UN AVOTI

[1] Valsts ieņēmumu dienests. Ziņojums par grūtībām pieslēgties EDS sistēmai. [tiešsaiste]. – [atsauce 10.05.2018.]. Pieejams: https://www.vid.gov.lv/lv/apgrutinata-pieslegsanas-vid-elektroniskas-deklaresanas-sistemai

[2] CERT. Informācijas tehnoloģiju drošības incidentu novērošanas institūcija. Ziņojums par grūtībām pieslēgties eVeselība sistēmai. [tiešsaiste]. – [atsauce 10.05.2018.]. Pieejams: https://cert.lv/lv/2018/01/cert-lv-e-veseliba-netika-uzlauzta

[3] Wiegers, K., & Beatty, J. (2013). Software requirements. Pearson Education.

[4] Young, R. R. (2004). The requirements engineering handbook. Artech House.

[5] Koelsch, G. (2016). Requirements Writing for System Engineering. Apress.

[6] Spillner, A., Linz, T., & Schaefer, H. (2014). Software testing foundations: a study guide for the certified tester exam. Rocky Nook, Inc..

[7] International Academy, Research, and Industry Association. Specifying Effective Non-Functional Requirements. [tiešsaiste]. – [atsauce 10.05.2018..]. Pieejams: https://www.iaria.org/conferences2012/filesICCGI12/Tutorial%20Specifying%20Effective%20Non-func.pdf

[8] Zane Bičevska "Viedās tehnoloģijas un to efektivitāte", promocijas darbs doktora disertācijai, LU Datorikas fakultātes, Latvijas Universitāte, 2010., pieejams https://dspace.lu.lv/dspace/handle/7/4619

[9] Pressman, R. S. (2005). Software engineering: a practitioner's approach. Palgrave Macmillan.

[10] Stack overflow. World's largest developer community. Discussion about difference between functional and nonfunctional requirements. [tiešsaiste]. – [atsauce 10.05.2018.]. Pieejams: https://stackoverflow.com/questions/16475979/what-is-the-difference-between-functional-and-non-functional-requirement

[11] TechTarget. Using a nonfunctional requirements template. [tiešsaiste]. – [atsauce 10.05.2018.]. Pieejams: http://searchsoftwarequality.techtarget.com/tip/Using-a-nonfunctional-requirements-template-plus-examples

[12] Sommerville, I. (2011). Software Engineering, 9th ed., Chapter 4. Addison-Wesley.

[13] Athens University of Economics and Business. Non-functional Requirement Metrics. [tiešsaiste]. – [atsauce 10.05.2018.]. Pieejams: https://www2.dmst.aueb.gr/dds/etech/swdev/nfmetric.htm

[14] Computer Science, University of Toronto. Lecture about non – functional requirements. [tiešsaiste]. – [atsauce 10.05.2018.]. Pieejams: http://www.cs.toronto.edu/~sme/CSC340F/slides/16-NFRs.pdf

[15] The University of Edinburgh. Non-functional requirements metrics. [tiešsaiste]. – [atsauce 10.05.2018.]. Pieejams: https://www.inf.ed.ac.uk/teaching/courses/inf2c-se/Lectures/2016/metrics.pdf

[16] University of Ottawa. Non-Functional requirements - qualities. [tiešsaiste]. – [atsauce 10.05.2018.]. Pieejams: http://www.eiti.uottawa.ca/~bochmann/SEG3101/Notes/SEG3101-ch3-4%20-%20Non-Functional%20Requirements%20-%20Qualities.pdf

[17] Latvijas Zinātņu Akadēmijas Terminoloģijas komisija. [tiešsaiste]. – [atsauce 10.05.2018.]. Pieejams: http://termini.lza.lv

[18] Molyneaux, I. (2009). The art of application performance testing: Help for programmers and quality assurance. " O'Reilly Media, Inc.".

[19] Microsoft Developer Network. Chapter 2 – Types of Performance Testing. [tiešsaiste]. – [atsauce 10.05.2018.]. Pieejams: https://msdn.microsoft.com/en-us/library/bb924357.aspx

[20] Stackify. The Ultimate Guide to Performance Testing and Software Testing [tiešsaiste]. – [atsauce 10.05.2018.]. Pieejams: https://stackify.com/ultimate-guide-performance-testing-and-software-testing/

[21] Wikipedia. Software performance testing. [tiešsaiste]. – [atsauce 10.05.2018.]. Pieejams: https://en.wikipedia.org/wiki/Software_performance_testing

[22] Kephart, J. O., & Chess, D. M. (2003). The vision of autonomic computing. Computer, 36(1), 41-50.

[23] Lalanda, P., McCann, J. A., & Diaconescu, A. (2013). Autonomic computing. Springer.

[24] University of St Andrews. The Autonomic Computing Vision. [tiešsaiste]. – [atsauce 10.05.2018.]. Pieejams: https://www.cs.st-andrews.ac.uk/files/2003-4-2%20Lecture1.pdf

[25] Edgars Diebelis "Programmatūras paštestēšana", promocijas darbs doktora disertācijai, LU Datorikas fakultātes, Latvijas Universitāte, 2012., pieejams http://dspace.lu.lv/dspace/handle/7/4708

[26] Kenyaplex. Factors affecting computer performance. [tiešsaiste]. – [atsauce 10.05.2018.]. Pieejams: https://www.kenyaplex.com/resources/5558-factors-affecting-computer-performance.aspx

[27] Software engineering. Logging into text file or database? [tiešsaiste]. – [atsauce 10.05.2018.]. Pieejams: https://softwareengineering.stackexchange.com/questions/313582/logging-into-text-file-or-database

[28] Ivo Odītis "Biznesa procesu izpildes laika verificēšana", promocijas darbs doktora disertācijai, LU Datorikas fakultātes, Latvijas Universitāte, 2016., pieejams http://dspace.lu.lv/dspace/handle/7/34336

PIELIKUMI

1. Pielikums. Analizējamā žurnālfaila piemērs

```
local,"587765e8a1cb6f37f7772ca2","589b2300a4f3820f57bbf856","/units","Thu May 03 2018
15:23:49 GMT+0000 (UTC)","Thu May 03 2018 15:23:51 GMT+0000 (UTC)","1766","Waiting"
local,"587765e8a1cb6f37f7772ca2","59e5b12dc204c20019b3a21f","/units","Thu May 03 2018
15:23:49 GMT+0000 (UTC)","Thu May 03 2018 15:23:51 GMT+0000 (UTC)","1773","Waiting"
local,"587765e8a1cb6f37f7772ca2","59e5b12dc204c20019b3a214","/units","Thu May 03 2018
15:23:49 GMT+0000 (UTC)","Thu May 03 2018 15:23:51 GMT+0000 (UTC)","1781","Waiting"
local,"587765e8a1cb6f37f7772ca2","589b2300a4f3820f57bbf856","/units","Thu May 03 2018
15:23:49 GMT+0000 (UTC)","Thu May 03 2018 15:23:51 GMT+0000 (UTC)","1774","Waiting"
local,"587765e8a1cb6f37f7772ca2","58c02d0437a4e40049a457e3","/units","Thu May 03 2018
15:23:49 GMT+0000 (UTC)","Thu May 03 2018 15:23:51 GMT+0000 (UTC)","2006","Completed"
local,"587765e8a1cb6f37f7772ca2","59e5b12dc204c20019b3a21f","/units","Thu May 03 2018
15:23:49 GMT+0000 (UTC)","Thu May 03 2018 15:23:51 GMT+0000 (UTC)","1785","Waiting"
local,"587765e8a1cb6f37f7772ca2","59e5b12dc204c20019b3a219","/units","Thu May 03 2018
15:23:49 GMT+0000 (UTC)","Thu May 03 2018 15:23:51 GMT+0000 (UTC)","2014","Completed"
local,"587765e8a1cb6f37f7772ca2","59e5b12dc204c20019b3a217","/units","Thu May 03 2018
15:23:49 GMT+0000 (UTC)","Thu May 03 2018 15:23:51 GMT+0000 (UTC)","2020","Completed"
local,"587765e8a1cb6f37f7772ca2","58c815beb4b65c010748628e","/units","Thu May 03 2018
15:23:50 GMT+0000 (UTC)","Thu May 03 2018 15:23:51 GMT+0000 (UTC)","1100","Waiting"
local,"587765e8a1cb6f37f7772ca2","58d3cc6aea8a1102668801e6","/units","Thu May 03 2018
15:23:49 GMT+0000 (UTC)","Thu May 03 2018 15:23:51 GMT+0000 (UTC)","2025","Completed"
local,"587765e8a1cb6f37f7772ca2","5a27efbeeb5bc70059f4e1c8","/units","Thu May 03 2018
15:23:50 GMT+0000 (UTC)","Thu May 03 2018 15:23:51 GMT+0000 (UTC)","1115","Waiting"
local,"587765e8a1cb6f37f7772ca2","58c815beb4b65c010748628e","/units","Thu May 03 2018
15:23:50 GMT+0000 (UTC)","Thu May 03 2018 15:23:51 GMT+0000 (UTC)","1140","Waiting"
local,"587765e8a1cb6f37f7772ca2","589b2300a4f3820f57bbf856","/units","Thu May 03 2018
15:23:50 GMT+0000 (UTC)","Thu May 03 2018 15:23:51 GMT+0000 (UTC)","1158","Waiting"
```

2. Pielikums. Kods

```
<?php
header('Content-type: text/html; charset=UTF-8') ;

 DEFINE('DB_USERNAME', 'root');
 DEFINE('DB_PASSWORD', 'root');
 DEFINE('DB_HOST', 'localhost');
 DEFINE('DB_DATABASE', 'Rindas1');
 $conn = new mysqli(DB_HOST, DB_USERNAME, DB_PASSWORD, DB_DATABASE);

 if (mysqli_connect_error()) {
  die('Connect Error ('.mysqli_connect_errno().') '.mysqli_connect_error());
 }

 echo 'Connected successfully.' . "<br>";

$row = 1;
        if (($handle = fopen("in1.csv", "r")) !== FALSE) {
   while (($data = fgetcsv($handle, 1000, ",")) !== FALSE) {
      $num = count($data);
      echo "<p> $num fields in line $row: <br /></p>\n";
      $row++;
      $sqlIns = "INSERT INTO logs (env, company, user, module, starttime, endtime, length)
values (";
      for ($c=0; $c < $num; $c++) {
         echo $data[$c] . " - ";
         if ($c < 4) {
                  $sqlIns = $sqlIns . "'" . $data[$c] . "', ";
                  }
               else if ($c == 6) {
                  $sqlIns = $sqlIns . "'" . $data[$c] . "') ";
               } else {
                        $data[$c] = substr($data[$c], 0, -6);
                                 $datums1 = DateTime::createFromFormat('D M d Y H:i:s
\\G\\M\\TO', $data[$c]);
                  $sqlIns = $sqlIns . "'" . $datums1->format('Y-m-d H:i:s') . "', ";
               }
      }
      echo "<p><br /> ". $sqlIns ." </p>";
                  if ($conn->query($sqlIns) === TRUE) {
                  echo "New record created successfully";
                  } else {
                  echo "Error: " . $sql . "<br>" . $conn->error;
                  }
```

```
    }
    echo "<hr/>Reading in done!";
    fclose($handle);
} else {
    echo "could not open csv";
}

?>
```

3. Pielikums. Loadtest saskarnes piemērs

```
MAC-KSN:2people ksn$ loadtest -c 25 -n 2000 -H "Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiI1OGQ1M2UyMDI2ZDI0ZTAxODA3MmI5NzMiLC
J1c3IiOiJBZG1pbiBBZG1pbm8iLCJlbWwiOiJhZG1pbkBlbWFpbC5jb20iLCJjaWQiOiI1ODc3NjVlOGExY2I
2ZjM3Zjc3NzJjYTIiLCJqdGkiOiI2M2U3ZTJjYy1iNTU5LTRjNGEtYTJkNi1hZjQzMmM1NzQ5MzkiLCJwZ
XIiOltdLCJpYXQiOjE1MjUzNTc5NDksImV4cCI6MTUyNTQ0NDM0OX0.KZmERGn8vXdiwK0UZTTlew
4MNxnV6av1TkhYS-kenhs" http://adevo.2peoplepro.com/api/v1/unit
headers: object, {"host":"adevo.2peoplepro.com","user-agent":"loadtest/3.0.3","accept":"*/*","authorization":"
Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiI1OGQ1M2UyMDI2ZDI0ZTAxODA3MmI5NzMiLC
J1c3IiOiJBZG1pbiBBZG1pbm8iLCJlbWwiOiJhZG1pbkBlbWFpbC5jb20iLCJjaWQiOiI1ODc3NjVlOGExY2I
2ZjM3Zjc3NzJjYTIiLCJqdGkiOiI2M2U3ZTJjYy1iNTU5LTRjNGEtYTJkNi1hZjQzMmM1NzQ5MzkiLCJwZ
XIiOltdLCJpYXQiOjE1MjUzNTc5NDksImV4cCI6MTUyNTQ0NDM0OX0.KZmERGn8vXdiwK0UZTTlew
4MNxnV6av1TkhYS-kenhs"}
[Thu May 03 2018 18:07:11 GMT+0300 (EEST)] INFO Requests: 0 (0%), requests per second: 0, mean
latency: 0 ms
[Thu May 03 2018 18:07:16 GMT+0300 (EEST)] INFO Requests: 238 (12%), requests per second: 48, mean
latency: 479.2 ms
[Thu May 03 2018 18:07:21 GMT+0300 (EEST)] INFO Requests: 467 (23%), requests per second: 46, mean
latency: 562.9 ms
[Thu May 03 2018 18:07:26 GMT+0300 (EEST)] INFO Requests: 735 (37%), requests per second: 54, mean
latency: 486.5 ms
[Thu May 03 2018 18:07:31 GMT+0300 (EEST)] INFO Requests: 1000 (50%), requests per second: 53, mean
latency: 467.4 ms
[Thu May 03 2018 18:07:36 GMT+0300 (EEST)] INFO Requests: 1198 (60%), requests per second: 40, mean
latency: 582.9 ms
[Thu May 03 2018 18:07:41 GMT+0300 (EEST)] INFO Requests: 1455 (73%), requests per second: 51, mean
latency: 508.2 ms
[Thu May 03 2018 18:07:46 GMT+0300 (EEST)] INFO Requests: 1720 (86%), requests per second: 53, mean
latency: 464.8 ms
[Thu May 03 2018 18:07:51 GMT+0300 (EEST)] INFO Requests: 1978 (99%), requests per second: 52, mean
latency: 483.7 ms
[Thu May 03 2018 18:07:51 GMT+0300 (EEST)] INFO
[Thu May 03 2018 18:07:51 GMT+0300 (EEST)] INFO Target URL:
http://adevo.2peoplepro.com/api/v1/unit
[Thu May 03 2018 18:07:51 GMT+0300 (EEST)] INFO Max requests:       2000
[Thu May 03 2018 18:07:51 GMT+0300 (EEST)] INFO Concurrency level:   25
[Thu May 03 2018 18:07:51 GMT+0300 (EEST)] INFO Agent:              none
[Thu May 03 2018 18:07:51 GMT+0300 (EEST)] INFO
[Thu May 03 2018 18:07:51 GMT+0300 (EEST)] INFO Completed requests:  2000
[Thu May 03 2018 18:07:51 GMT+0300 (EEST)] INFO Total errors:        0
[Thu May 03 2018 18:07:51 GMT+0300 (EEST)] INFO Total time:          40.280555329 s
[Thu May 03 2018 18:07:51 GMT+0300 (EEST)] INFO Requests per second: 50
[Thu May 03 2018 18:07:51 GMT+0300 (EEST)] INFO Mean latency:        501.3 ms
[Thu May 03 2018 18:07:51 GMT+0300 (EEST)] INFO
[Thu May 03 2018 18:07:51 GMT+0300 (EEST)] INFO Percentage of the requests served within a certain time
[Thu May 03 2018 18:07:51 GMT+0300 (EEST)] INFO   50%      493 ms
[Thu May 03 2018 18:07:51 GMT+0300 (EEST)] INFO   90%      636 ms
[Thu May 03 2018 18:07:51 GMT+0300 (EEST)] INFO   95%      674 ms
[Thu May 03 2018 18:07:51 GMT+0300 (EEST)] INFO   99%      1588 ms
[Thu May 03 2018 18:07:51 GMT+0300 (EEST)] INFO  100%      1910 ms (longest request)
```

4. Pielikums. Slodzes testa datu kopsavilkums

Pieprasījumu skaits	Atļautie lietotāji	Paralēlie pieprasījumi pakotnē	Statuss	Vidējais atbildes laiks	Sākuma laiks	Beigu laiks	Kopējais apstrādes ilgums
105	6%	10	Completed	152.1048	03.05.18 15:41	03.05.18 15:42	27
131	6%	100	Completed	956.3664	03.05.18 15:43	03.05.18 15:43	20
153	6%	250	Completed	2307.6667	03.05.18 15:44	03.05.18 15:44	20
144	6%	500	Completed	4472.8750	03.05.18 15:45	03.05.18 15:45	24
130	6%	50	Completed	470.9692	03.05.18 15:45	03.05.18 15:46	19
125	6%	25	Completed	259.1280	03.05.18 15:46	03.05.18 15:46	20
148	6%	750	Completed	6320.4324	03.05.18 15:47	03.05.18 15:47	24
1904	6%	10	Waiting	108.5667	03.05.18 15:41	03.05.18 15:42	28
1967	6%	100	Waiting	556.5099	03.05.18 15:43	03.05.18 15:43	19
1947	6%	250	Waiting	1225.1495	03.05.18 15:44	03.05.18 15:44	19
2279	6%	500	Waiting	2386.7679	03.05.18 15:45	03.05.18 15:45	24
1919	6%	50	Waiting	280.5836	03.05.18 15:45	03.05.18 15:46	19
1899	6%	25	Waiting	157.2670	03.05.18 15:46	03.05.18 15:46	20
2286	6%	750	Waiting	3383.0512	03.05.18 15:47	03.05.18 15:47	24
608	29%	10	Completed	166.1891	03.05.18 15:27	03.05.18 15:27	31
487	29%	25	Completed	355.2094	03.05.18 15:28	03.05.18 15:28	20
602	29%	50	Completed	639.0382	03.05.18 15:29	03.05.18 15:30	24
664	29%	100	Completed	1130.3404	03.05.18 15:30	03.05.18 15:30	23
699	29%	250	Completed	2538.9671	03.05.18 15:31	03.05.18 15:32	25
778	29%	500	Completed	4983.9987	03.05.18 15:32	03.05.18 15:33	28
787	29%	750	Completed	7897.5362	03.05.18 15:34	03.05.18 15:34	32
1401	29%	10	Waiting	109.4183	03.05.18 15:27	03.05.18 15:27	31
1174	29%	25	Waiting	156.1601	03.05.18 15:28	03.05.18 15:28	20
1447	29%	50	Waiting	280.4713	03.05.18 15:29	03.05.18 15:30	24
1435	29%	100	Waiting	475.4871	03.05.18 15:30	03.05.18 15:30	22
1550	29%	250	Waiting	1195.6271	03.05.18 15:31	03.05.18 15:32	26
1701	29%	500	Waiting	2350.9077	03.05.18 15:32	03.05.18 15:33	27
1754	29%	750	Waiting	3389.2429	03.05.18 15:34	03.05.18 15:34	31
1216	59%	10	Completed	185.5288	03.05.18 15:16	03.05.18 15:16	37
1220	59%	25	Completed	400.1631	03.05.18 15:17	03.05.18 15:18	30
1212	59%	50	Completed	799.8828	03.05.18 15:18	03.05.18 15:18	29
1296	59%	100	Completed	1546.9120	03.05.18 15:20	03.05.18 15:20	29
1395	59%	250	Completed	3640.7269	03.05.18 15:21	03.05.18 15:21	30
1529	59%	500	Completed	7670.1812	03.05.18 15:22	03.05.18 15:22	37
1503	59%	750	Completed	10243.9701	03.05.18 15:23	03.05.18 15:24	37
793	59%	10	Waiting	114.3569	03.05.18 15:16	03.05.18 15:16	37
804	59%	25	Waiting	143.0697	03.05.18 15:17	03.05.18 15:18	30

837	59%	50	Waiting	214.2712	03.05.18 15:18	03.05.18 15:18	29
803	59%	100	Waiting	358.8966	03.05.18 15:20	03.05.18 15:20	29
854	59%	250	Waiting	745.8864	03.05.18 15:21	03.05.18 15:21	28
955	59%	500	Waiting	1428.3497	03.05.18 15:22	03.05.18 15:22	35
1098	59%	750	Waiting	2450.1430	03.05.18 15:23	03.05.18 15:24	33
2009	100%	10	Completed	195.9915	03.05.18 15:07	03.05.18 15:08	44
2024	100%	25	Completed	439.0385	03.05.18 15:08	03.05.18 15:09	41
2049	100%	50	Completed	772.5564	03.05.18 15:09	03.05.18 15:10	37
2099	100%	100	Completed	1542.0872	03.05.18 15:10	03.05.18 15:11	39
2248	100%	250	Completed	3426.9159	03.05.18 15:11	03.05.18 15:11	38
2447	100%	500	Completed	7053.8913	03.05.18 15:12	03.05.18 15:12	45
2705	100%	750	Completed	10894.8399	03.05.18 15:13	03.05.18 15:14	50

DOKUMENTĀRĀ LAPA

Maģistra darbs "Pašregulējoša aizsardzība pret sistēmas pārslodzi" izstrādāts LU Datorikas fakultātē.

Darba teksta galīgā versija izgatavota 21.05.2018

Ar savu parakstu apliecinu, ka pētījums veikts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti un iesniegtā darba elektroniskā kopija atbilst izdrukai.

Autors: _____

(Autora paraksts un datums)

Ar savu parakstu apliecinu, ka esmu lasījis augstāk minēto maģistra darbu un atzīstu to par **p i e m ē r o t u / n e p i e m ē r o t u** (nevajadzīgo svītrot) aizstāvēšanai Latvijas Universitātes datorzinātņu maģistrantūrā.

Darba vadītājs: _____

(Vadītāja paraksts un datums)

Darbs iesniegts maģistratūras sekretariātā _____.

(Iesniegšanas datums)

Ar šo es apliecinu, ka darba elektroniskā versija ir augšupielādēta LU informatīvajā sistēmā.

Studiju metodiķe: _____.

(Metodiķes paraksts)

Recenzents: docents Dr.dat. Leo Trukšāns _____

(Akad.amats, zin.grāds, vārds, uzvārds)

Darbs aizstāvēts maģistra gala pārbaudījuma komisijas sēdē

_____ prot. Nr. _____

(Darba aizstāvēšanas datums)

Komisijas sekretārs: _____

(Sekretāra paraksts)