TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatika instituut
Infosüsteemide õppetool

Andmete logimise ja presenteerimise süsteemi optimeerimine

Bakalaureusetöö

Üliõpilane: Karmo Kuurberg
Üliõpilaskood: 120941IAPB
Juhendaja: Raul Liivrand

Tallinn
2015

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

(*kuupäev*) (*allkiri*)

Annotatsioon

Andmete logimise ja presenteerimise süsteemi optimeerimine

Käesoleva töö eesmärk on optimeerida olemasolevat andmete logimise süsteemi nii, et see kasutaks vähem andmemahtu ja võimaldaks teha kiiremaid päringuid ülevaatliku graafiku kuvamiseks, vähendades ooteaega minutitest sekunditele.

Töös otsitakse andmemahu säästmiseks lahendust probleemile, kuidas kirjutada andmebaasi logitavate väärtuste muutusi nii, et ei peaks kordama mitte muutunud väärtusi, kuid samas mitte andes järele kiirusenõuetes. Lugemise kiirendamiseks pakutakse välja võimalus teha logimise ajal andmetest vahekokkuvõtteid ja realiseeritakse see.

Töö tulemusena valmib logiandmete kirjutaja ja andmebaasisüsteemi vahele vahekiht, mis töötleb andmed sobivale kujule, et realiseerida eelpool mainitud eesmärgid.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 47 leheküljel, 6 peatükki, 17 joonist, 16 tabelit.

Abstract

Optimizing the system of logging and presenting data

The aim of this work is to optimize an existing system of logging data, so that it would use less disk space and would support faster queries for presenting an overview graph, reducing the wait time from minutes to seconds.

In this work, it is explored how to write logged data to the database in a way that it would not be necessary to repeat values which did not change while keeping the speed requirements. In order to speed up reading, an option to summarize data while logging will be offered and implemented.

As the result of this study, an intermediate layer will be added between the logger and the database management system. Its purpose is to process the data into a form necessary to fulfill previously mentioned aims.

The thesis is in Estonian and contains 47 pages of text, 6 chapters, 17 figures, 16 tables.

Lühendite ja mõistete sõnastik

bloob	***BLOB (Binary Large OBject)*** Andmebaasi salvestatud suur bitiplokk (pilt, helifail, videoklipp, mõnikord ka binaarkood), mida andmebaasihaldur ise ei interpreteeri. [1]
CDP	***Control Design Platform*** Ettevõtte ICD Software toodetud rakendusraamistik juhtsüsteemide loomiseks. [2]
CDP signaal	***CDP signal*** CDP rakendusraamistiku kaudu teisele komponendile (nt andmete logimise komponendile) edastatud numbriline väärtus. Tavaliselt pärineb info mingisuguselt andurilt, kuid tegu võib olla ka näiteks süsteemi mälukasutuse või protsessori hõivatusega.
signaal	***signal*** Selles töös mõeldakse signaali all CDP signaalilt pärit infot ehk siis mingisugust numbrilist väärtust.
API	***Application Programming Interface*** Rakendusliides. Reeglistik olemasoleva valmisprogrammiga suhtlemiseks. [3]
WAL	***Write-Ahead Logging*** Tehnika, mille korral andmebaasi tehtavad muudatused salvestatakse esmalt põhiandmete failist eraldi logifaili. Seetõttu pole andmete kirjutamiseks vaja andmebaasist lugemist lukustada. [4]

Jooniste nimekiri

Tabelite nimekiri

Sisukord

1. Sissejuhatus

On tavaline, et seadmetes on mitmesuguseid andureid, mis toodavad suurel hulgal andmeid. Kogutud informatsioon tuleb salvestada ja mõnda aega alles hoida, et jälgida seadme tööd ning vea tekkimisel leida selle põhjus.

Olemasoleval logimise süsteemil on kaks põhilist ülesannet: esiteks valitud signaalide väärtuste logimine ja teiseks nende kuvamine graafikul, kus ühel teljel on aeg ja teisel signaali väärtus sel hetkel.

Põhilisi probleeme on samuti kaks. Esiteks liigne andmemahu kasutus: signaale logitakse perioodiliselt ning kui ühe väärtus muutub, salvestatakse kõikide väärtus hoolimata sellest, et teised ei muutunud. Teiseks päringute aeglus: kui andmeid on gigabaitides, siis võtab ülevaatliku graafiku kuvamine aega kümneid sekundeid või isegi minuteid.

Töö tehti kahes osas – 2014. aasta suve lõpus ja 2015. aasta alguses – ettevõtte ICD Industries Estonia OÜ tarbeks ning see võetakse tõenäoliselt kasutusele laevadel andmete logimiseks ja kuvamiseks.

1.1 Ülesande püstitus

Ülesandeks on optimeerida olemasolevat logimise süsteemi nii, et:

- kasutataks vähem andmemahtu eeldusel, et enamus logitavaid signaale muutuvad harva;
- päringute tegemine ja nende põhjal ülevaatliku graafiku kuvamine oleks oluliselt kiirem, mitte üle paari sekundi.

Töö eesmärgiks on luua logiandmeid vastu võtva komponendi jaoks vahekiht, mis teisendaks andmed vajalikku formaati, et täita ülal mainitud nõudeid, ja seejärel salvestaks need andmebaasi.

1.2 Ülevaade tööst

Analüüsi peatükis esmalt kirjeldatakse olemasolevat logimise süsteemi ja tuuakse välja selle puudused. Seejärel katsetatakse ja võrreldakse võimalikke lahendusvariante.

Tehnilise lahenduse peatükis seletatakse, kuidas uus logimise süsteem realiseeriti. Esmalt kirjeldatakse kasutatavaid tehnoloogiaid ja andmebaasi struktuuri, järgneb kirjutamise ja lugemise API ning realisatsiooni kirjeldus.

Töö lõpus testitakse uut lahendust ja võrreldakse seda olemasolevaga. Vaadatakse lugemis- ja kirjutamiskiirust ning kasutatavat andmemahtu.

2. Analüüs

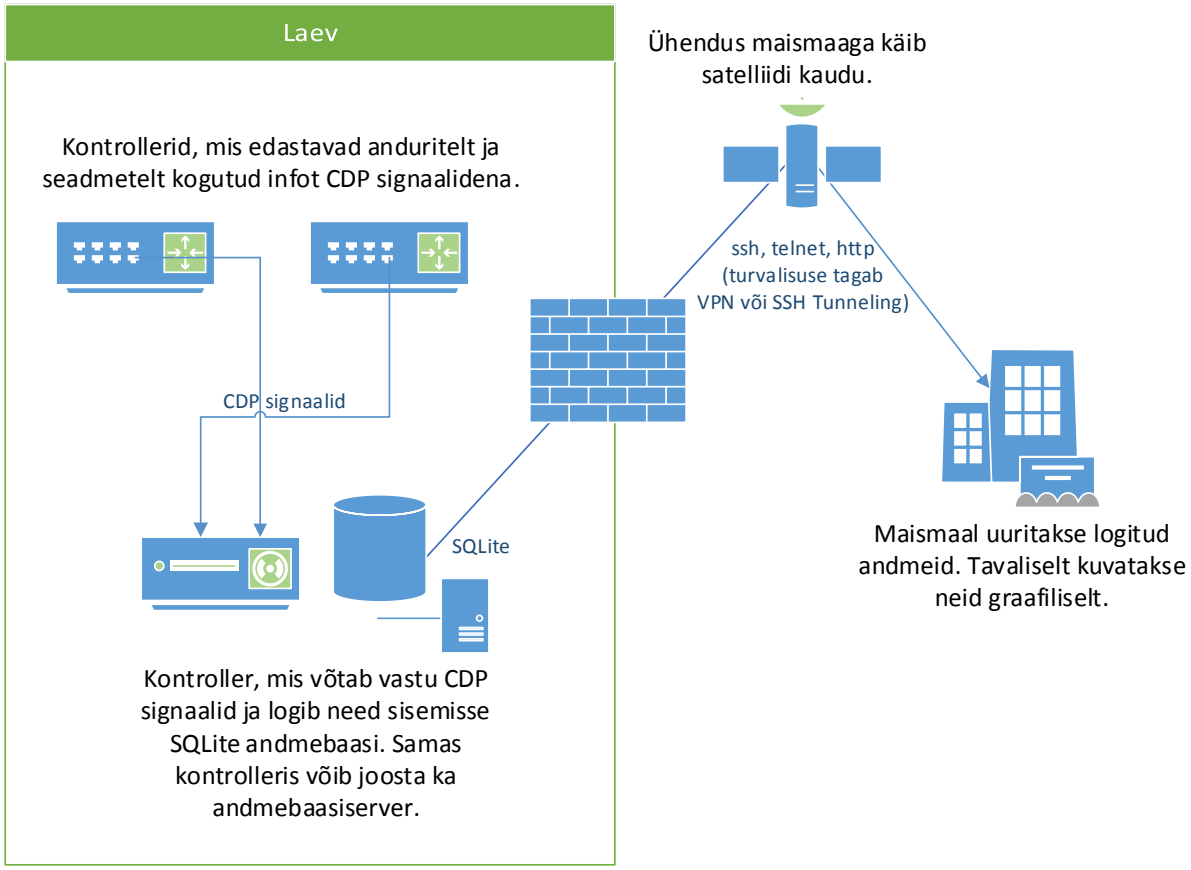
2.1 Olemasolev lahendus

2.1.1 Taust

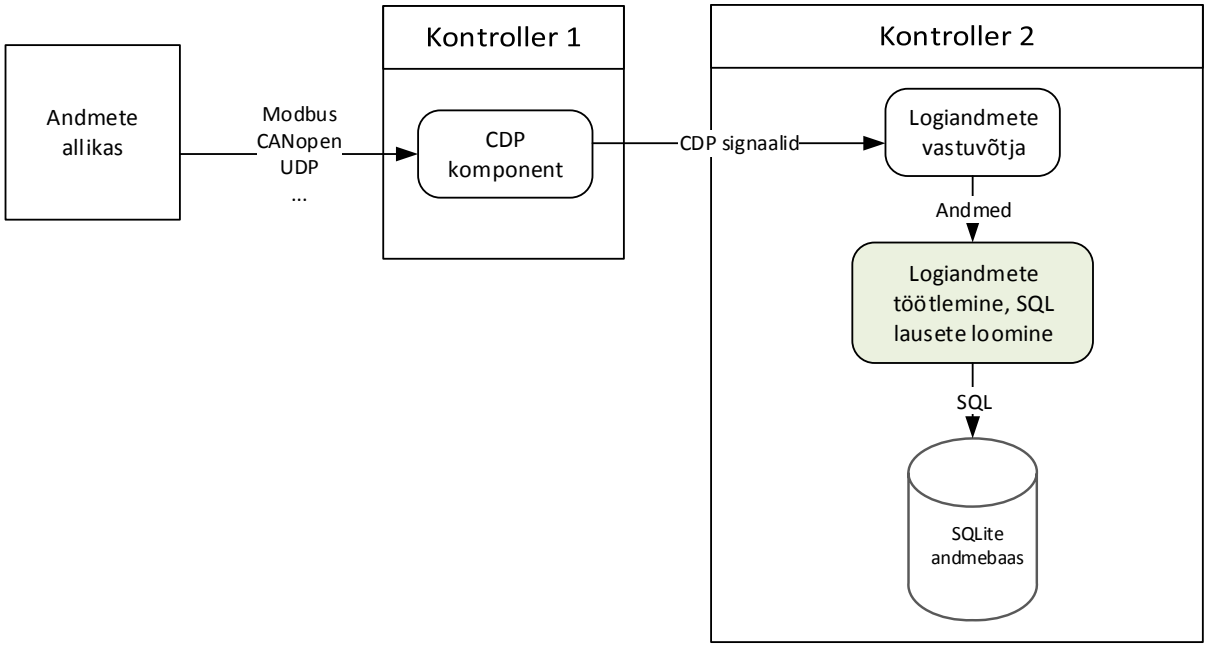
Olemasolevat logimise süsteemi kasutatakse põhiliselt laevadel. Sealsete seadmete andurid toodavad ohtralt andmeid, mis edastatakse andmeid logivale komponendile CDP rakendusraamistiku signaalide abil (vt CDP ja CDP signaal, lk 5).

Andmeid logiv komponent ise asub tavaliselt teistest seadmetest eraldi, oma kontrolleri peal. Andmekandjana kasutatakse üldiselt SSD-d, sest laevade kõikumine segab tavalise kõvaketta tööd.

Kui mõne seadme töös esineb viga, siis võib olla vajalik, et mõni spetsialist kalda peal vaataks logisid. Arvestada tuleb, et võrguühenduse kiirus satelliidi kaudu võib olla aeglane.



Joonis 1 Süsteemi arhitektuur.

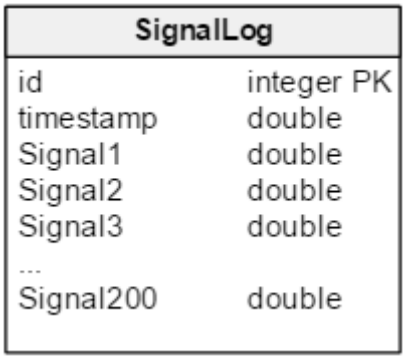


Joonis 2 Andmete liikumine süsteemis.

Käesolev töö puudutab ülal oleval joonisel rohelise varjundiga märgitud kasti.

2.1.2 Andmete logimine

Andmed logitakse lokaalsesse SQLite andmebaasi faili, kuhu on loodud iga signaaligrupi kohta üks suur tabel.



Joonis 3 Andmebaasi struktuur olemasolevas süsteemis.

	id	timestamp	GW1OilPress1HPU	GW1N2CounterPress1	GW1CylActivPress1
	Filt...	Filter	Filter	Filter	Filter
51	51	1338907424.232	0.0	70.7865168539326	0.146555935515388
52	52	1338907424.332	0.0488519785051295	71.0796287249634	0.244259892525647
53	53	1338907424.432	0.0	70.9819247679531	0.244259892525647

Joonis 4 Näide logitud andmetest olemasolevas süsteemis.

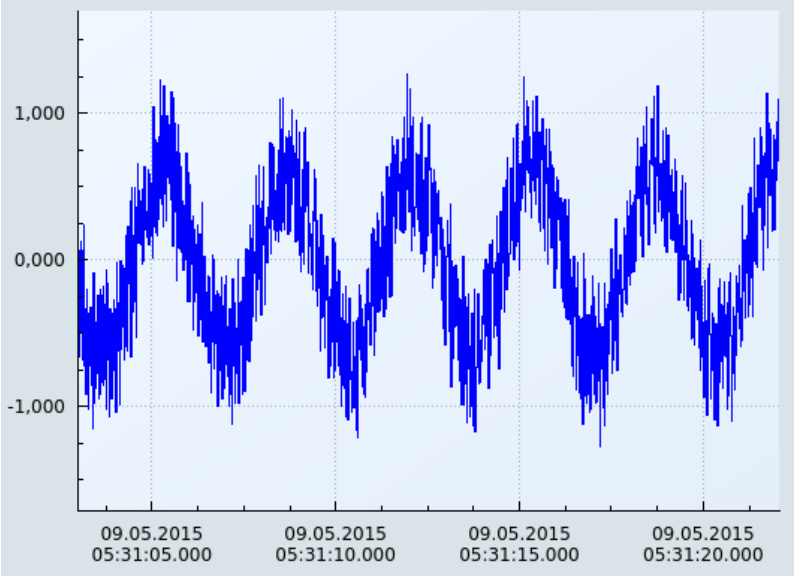
Tabeli igas reas hoitakse ajahetke ning iga signaali väärtust sel hetkel. Andmemahu kokku hoidmiseks on igale signaalile antud lisaparameeter, mis määrab, kui palju peab signaali väärtus muutuma selleks, et logitaks uus rida tabelisse. Indekseeritud veergu „id“ kasutatakse tabeliridade arvu piiramiseks – kui ridu on piisavalt, hakatakse UPDATE lausete abil algust üle kirjutama.

Lahenduse suurimaks puuduseks on see, et kui muutub ainult ühe signaali väärtus, siis andmebaasi rida lisades tuleb kirjutada väärtus igasse veergu. On selge, et mitte muutunud väärtuste kordamine kulutab asjatult andmemahtu.

Olukorda leevendab veidi see, et on võimalik teha signaaligruppe sarnase tihedusega muutuvatele signaalidele ning iga grupi jaoks luuakse oma tabel. Siiski iga signaali jaoks oma tabeli tegemine poleks samuti hea variant, sest tuleb arvestada, et igas tabelis peab olema veerg ajahetkele ja primaarvõti „id“, mis mõlemad võtavad ruumi.

2.1.3 Andmete lugemine

Logitud andmeid kuvatakse kasutajale graafiliselt.



Joonis 5 Graafiku kuvamine kasutajale.

Oletame, et andmebaasis on 100 000 rida ja graafiku laius on 1000 pikslit. Sel juhul tuleks kuvada ainult iga sajas rida. Selleks, et signaalide minimaalsed ja maksimaalsed väärtused kaduma ei läheks, kasutatakse SQL päringut, mis grupeerib read saja kaupa ning leiab igas grupis iga signaali jaos minimaalse ja maksimaalse väärtuse.

```
SELECT MAX(Timestamp), MIN(Signal1), MAX(Signal1)
FROM SignalLog
WHERE Timestamp > 100 AND Timestamp < 230
GROUP BY (CAST(Timestamp/(100000/1000) AS INT))
ORDER BY MAX(Timestamp);
```

Ajahetke veerule „Timestamp“ on loodud indeks, millest on kasu juhul, kui kasutaja otsustab ühte graafikulõiku suurendada, sest sel juhul saab läbi skaneerida ainult piirkonda jäävad read ja päring täidetakse oluliselt kiiremini.

Kahjuks pole indeksist kasu kogu määramispiirkonna kuvamisel. Sel juhul tuleb skaneerida tervet tabelit, sest see on ainus võimalus leidmaks igale ridade grupile signaalide minimaalsed ja maksimaalsed väärtused. Kui andmeid on gigabaitides, siis võib see aega võtta kümneid sekundeid või isegi minuteid.

Kui lugemiseks kasutatakse lokaalset andmebaasi, on võimalik ja soovitatav laadida kõikide graafikul kuvatavate signaalide jaoks andmebaasi sisu mällu (serveri puhul võtaks see liiga palju aega, sest võrguühendus laevadega üle satelliidi on aeglane). Sel juhul kulub pärast esmast laadimist graafiku suurendamiseks või liigutamiseks halvimal juhul kuni 5 sekundit, rohkem sisse suurendades töödeldava info maht väheneb ja kiirus paraneb.

2.2 Nõuded uuele süsteemile

2.2.1 Funktsionaalsed

Säilib olemasoleva süsteemi funktsionaalsus:

1. Signaalide väärtuste logimine koos ajahetkega.
2. Logitud andmete kuvamine graafikul.
3. Andmebaasi suuruse piiramine vanemate kirjete kustutamise abil.

2.2.2 Mittefunktsionaalsed

1. Logimise kiirus vähemalt 1000 korda sekundis 200 signaali korral.
2. Graafiku esialgse ülevaate kuvamine ega suurendamine ei võta aega üle kahe sekundi, isegi siis kui logitud andmeid on gigabaitides.

3. Graafikut võidakse kuvada üle võrgu. Et ühendus laevadega läbi satelliidi võib olla aeglane, peaks edastatav andmemaht olema võimalikult väike. Ei sobi lahendus, mis vajab terve andmebaasi alla tõmbamist ja mällu laadimist.

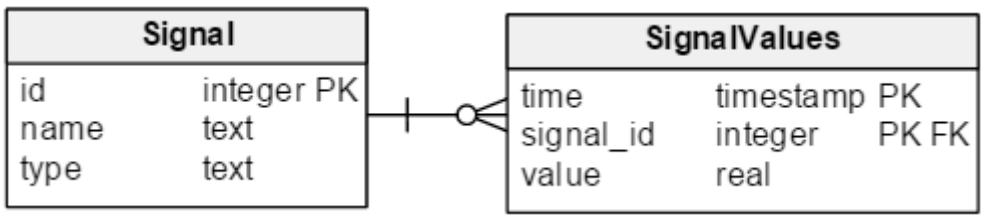
2.3 Signaalimuutuste hoidmise lahenduste võrdlus

Selles peatükis kirjeldatakse erinevaid lahendusvõimalusi, testitakse nende kiirust ja hinnatakse nende sobivust. Vaadates kiirusetestide vastavust eelpool toodud mittefunktsionaalsetele nõuetele, tuleks kindlasti arvestada, et riistvara, mille peal loodav süsteem kasutusele võetakse, on nõrgema jõudlusega kui testsüsteem.

Kõiki kiiruseteste jooksutati kolm korda ja tulemuseks võeti nende keskmine. Iga testi prooviti SQLite peal, mida kasutab olemasolev süsteem, ning võrdluseks ka suuremahulisemal ja võimalusterohkemal PostgreSQL andmebaasisüsteemil, mis vajab eraldi protsessis töötavat serverit [5].

Et oleks lihtne vahetada andmebaasisüsteemide vahel, kasutati ühenduse loomiseks Qt rakendusraamistiku Qt SQL teeki, mille abil saab läbi ühise liidese kasutada erinevaid andmebaasidraivereid [6]. SQLite ja PostgreSQL süsteemide arusaam SQL keelest olid piisavalt sarnased, et mõlema korral sai jooksutada sama koodi.

2.3.1 Üks rida, üks signaalimuutus



Joonis 6 Igal tabeli real saab olla ainult ühe signaali väärtuse muutus.

Eelised võrreldes olemasoleva süsteemiga:

- Kui muutub ainult üks signaal, siis kirjutatakse andmebaasitabeli reale ainult selle signaali ID ja uus väärtus. Kuna igale signaalile pole oma veergu, siis mitte muutunud väärtusi kordama ei pea. Harva muutuvate signaalide korral säästab see andmemahtu.

- Uusi signaale saab lisada ilma, et peaks andmebaasitabelile veerge lisama.

- Kui kaob vajadus logida ühte signaali, siis ei pea kordama selle väärtust igal real.

Puudused:

- Kui kõik signaalid iga logimise takt muutuvad, siis tuleb mitu korda korrata logimise aega (veerg „time“). See aga kulutab andmemahtu.
- Tabelisse tekib rohkem ridu:
    - Primaarvõtme jaoks loodud indeks muutub suuremaks ja selle uuendamine aeglasemaks.
    - Ühe suure rea lisamine on kiirem kui mitme väikse.

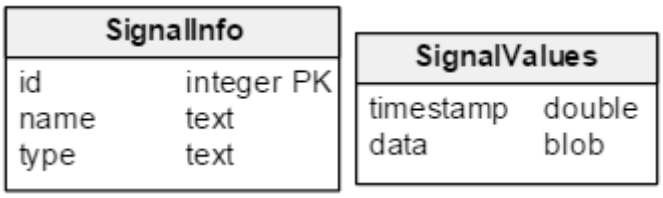
Lahenduse võimaliku aegluse tõttu tegin võrdlevad jõudlustestid järgmiste parameetritega: 200 signaali, iga takt kõik muutusid. Tulemused on kolme testi keskmised väärtused.

Tabel 1 Lahenduste võrdlus. Kirjutamise test.

	Olemasolev lahendus		Üks rida, üks signaalimuutus	
Andmebaasisüsteem	SQLite	PostgreSQL	SQLite	PostgreSQL
Ridu andmebaasis	15 000		3 000 000 (15 000 muutust × 200 signaali)	
Andmebaasi suurus	30,9 MB	13 MB	152,2 MB	240 MB
Kulunud aeg (sekundit)	3,475	25,50	67,19	552,8
Logimise sagedus (kordi sekundis)	4317	588	223	27,1

Nagu näha, siis pakutav lahendus on liiga aeglane ja ei vasta mittefunktsionaalsele nõudele 1 (vt lk 15), et logida peaks saama vähemalt 1000 korda sekundis. Muret teeb ka andmekasutus halvimal juhul, kui iga logimise takt kõikide signaalide väärtused muutusid.

2.3.2 Signaalimuutused bloobina



Joonis 7 Signaaliväärtuste muudatused kodeerituna bloobi.

Lahenduse mõte seisneb selles, et iga logimise takt kodeeritakse kõik muutunud signaaliväärtused bloobi sisse ja kirjutatakse ühe tabelireana andmebaasi.

Bloobi sisuks oleks nimekiri signaali ID ja väärtuse paaridest. Iga ID võtaks kaks baiti, väärtus vastavalt andmetüübi suurusele, näiteks *char* ühe, *double* kaheksa baiti. Ehk siis halvimal juhul kuluks ühe signaalimuutuse jaoks bloobis 10 baiti.

Sellel lahendusel on täpselt samad eelised, mis punktis 2.3.1 (vt lk 16, „Üks rida, üks signaalimuutus“) välja toodud lahendusel. Ehk siis logitakse ainult signaalide muutusi, säästes andmemahtu, ning uute signaalide lisamiseks ei ole vaja veerge lisada.

Puudused võrreldes olemasoleva lahendusega:

- Kuna andmeid hoitakse bloobina, ei saa nende töötlemiseks kasutada SQL keele deklaratiivseid võimalusi.
- Andmete logimiseks ja lugemiseks tuleb kirjutada protseduuriline kood, mis teisendab andmed bloobiks ja sealt tagasi.

Testisin ka seda lahendust 200 signaaliga:

Tabel 2 Signaalimuutused bloobina – kirjutamise test.

Andmebaasisüsteem	SQLite	PostgreSQL	SQLite	PostgreSQL
Igal real muutunud signaale	100%		20%	
Ridu andmebaasis	15 000			
Andmebaasi suurus	32,7 MB	41 MB	7,8 MB	6,7 MB
Kulunud aeg (sekundit)	2,098	5,233	0,808	2,275
Logimise sagedus (kordi sekundis)	7150	2866	18 560	6593

Nagu näha, siis see lahendus oleks isegi kiirem kui olemasolev lahendus, ka juhul kui kõik signaalid muutuvad iga takt. Lisaks sellele on oluline ruumisääst, kui osa signaale iga rida ei muutu: SQLite puhul näiteks vähenes andmemahu kasutus 4,2 korda, kui muutus ainult viiendik signaale.

Et teine pool ülesandest on lugemise kiiremaks tegemine, võrdlesin ka seda olemasoleva lahendusega. Päringud tehti ajahetke veeru järgi, millele oli loodud ka indeks.

Tabel 3 Lugemiskiiruse test. Loeti kõigi 200 signaali väärtused.

	Olemasolev lahendus		Signaalimuutused bloobina	
Andmebaasisüsteem	SQLite	PostgreSQL	SQLite	PostgreSQL
Ridu andmebaasis	15 000		15 000	
Andmebaasi suurus	30,9 MB	13 MB	32,7 MB	41 MB
Kulunud aeg (sekundit)	4,00	30,4	0,172	7,14

Tulemustest on näha, et mõlema andmebaasisüsteemi, kuid eriti SQLite puhul, on bloobina andmete lugemine kiirem, kui olemasoleva lahenduse tabelist.

Veel tuleb arvestada, et kui kasutaja soovib korraga ainult ühe signaali andmeid, siis bloobi puhul tuleks ikka lugeda andmebaasist kogu bloob, olemasoleva lahenduse korral saaks aga pärida ainult ühte veergu. Seega tegin testid ka ainult ühe signaali lugemise kohta.

Tabel 4 Lugemiskiiruse test. Loeti kõikide signaalide väärtused.

	Olemasolev lahendus	
Andmebaasisüsteem	SQLite	PostgreSQL
Ridu andmebaasis	15 000	
Andmebaasi suurus	30,9 MB	13 MB
Kulunud aeg (sekundit)	0,080	2,83

Kuna tavaliselt logides ei muutu kõik signaalid korraga, siis test ka selle kohta, kui kaua võtab bloobi lugemine aega, juhul kui igal real muutus ainult 20% väärtustest.

Tabel 5 Lugemiskiiruse test tabelist, mille igal real muutus 20% signaalidest.

	Signaalimuutused bloobina	
Andmebaasisüsteem	SQLite	PostgreSQL
Ridu andmebaasis	15 000	
Andmebaasi suurus	7,8 MB	6,7 MB
Kulunud aeg (sekundit)	0,114	2,85

Testi tulemused näitavad, et vana lahenduse puhul on andmebaasist ainult ühe veeru lugemine oluliselt kiirem kõigi korraga vaatamisest, samas vahe terve bloobirea lugemisega ei ole väga suur, eriti kui kõik signaalid alati logimise ajal ei muutunud. Siiski kuna testis ei ole arvestatud bloobi käsitsi lahtikodeerimisele kulunud aega, siis päris süsteemis võivad vahed suuremad olla. Sellegipoolest pole põhjust arvata, et see asja oluliselt aeglasemaks teeb, ja arvestades ka, et bloobide kasutamine võimaldaks olulist andemahu säästu, mis oli nimetatud

mittefunktsionaalsetes nõuetes (lk 15), siis otsustasin uue süsteemi loomisel kasutada ikkagi seda lahendust.

2.4 Andmete lugemise kiirendamine

2.4.1 Aeg-ajalt kõikide signaalide hetkeväärtuste kirjutamine

Kui salvestada andmebaasi ainult signaaliväärtuste muutused, siis tekib lugemisel üks probleem. Nimelt peab signaali väärtuse kättesaamiseks päritud ajahetkel otsima minevikust selle signaali viimase muutuse. Harva muutuvate signaalide korral võib see aga väga kaua aega võtta. Seega on mõistlik teha aeg-ajalt kokkuvõtteid ehk siis kirjutada andmebaasi vahetevahel kõikide signaalide väärtused.

2.4.2 Vahekokkuvõtete tegemine

Nagu mainitud punktis 2.1.3 (vt lk 14), teeb graafikul algse pildi kuvamise aeglaseks nõue, et hoolimata suurenduse astmest oleks näha signaalide minimaalsed ja maksimaalsed väärtused. Seega ei piisa vaid andmebaasist näiteks iga sajanda rea lugemisest, vaid tuleb läbi skaneerida kogu määramispiirkond.

Üks võimalik lahendus oleks teha logimise ajal aeg-ajalt vahekokkuvõtteid. Ehk siis salvestada, et mingist kellajast mingi kellaajani olid signaalide miinimumid ja maksimumid sellised. Nii peaks ülevaatliku graafiku näitamiseks lugema ainult seda kokkuvõttetabelit. Et lugemine oleks kiire igal suurendusastmel, võiks olla mitu kokkuvõttetabelit eri resolutsioonidega.

Keyframes0	
timestamp	double N
Signal1Min	int
Signal1Max	int
Signal1Last	int
Signal2Min	bigint
...	
Signal200Last	double

Keyframes1	
timestamp	double N
Signal1Min	int
Signal1Max	int
Signal1Last	int
Signal2Min	bigint
...	
Signal200Last	double

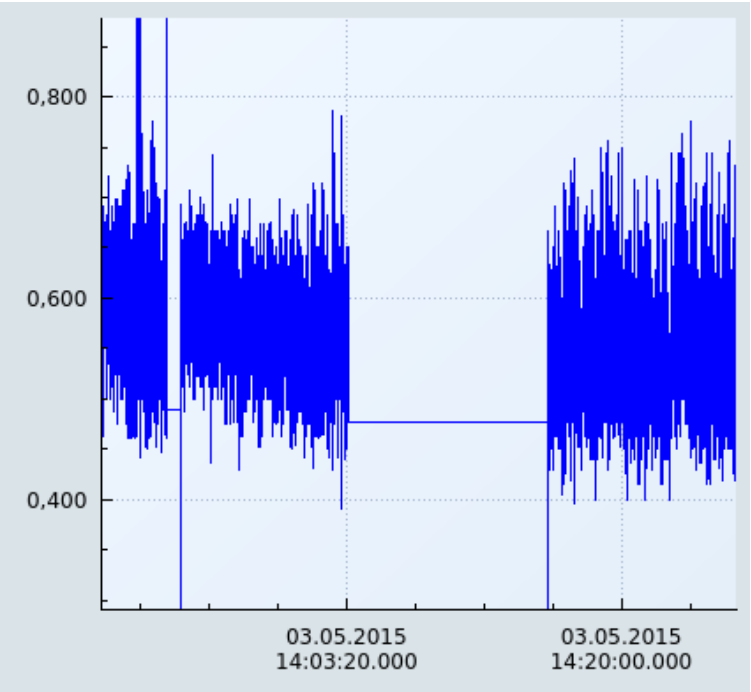
Joonis 8 Kokkuvõttetabelite struktuur.

Nagu ülal olevalt jooniselt näha, on kokkuvõttetabelis iga signaali jaoks 3 veergu: miinimum, maksimum ja viimane väärtus. Mõned tähelepanekud sellise valiku jaoks:

- Kuna iga signaali jaoks on oma veerg, siis tuleb korrata ka mitte muutunud väärtusi. Nii otsustasin teha punktis 2.4.1 (vt lk 21) kirjeldatud põhjustel lugemise kiirendamiseks.

- Miinimum ja maksimum on vajalik, et kuvada signaali väärtuste vahemik ülevaatlikul graafikul – sellisel, mis pole lõpuni sisse suurendatud.

- Viimane väärtus on selleks, et kui ükski signaal ei muutu, siis pole ka mõtet kokkuvõttetabelitesse kogu aeg kirjutada, et iga signaali miinimum ja maksimum on võrdsed ning sama, mis eelmine väärtus. Kui olemas on ka viimane väärtus, siis saab selle järgi tõmmata sirge joone kuni järgmise signaali muutuseni (vt Joonis 9).



Joonis 9 Viimase väärtuse abil tõmmatakse sirge joon õigele kõrgusele, kui signaal ei muutunud.

3. Tehniline lahendus

3.1 Lahenduse idee

3.1.1 Lühikirjeldus

Andmemahu kokkuhoidmiseks hoitakse andmebaasis ainult signaaliväärtuste muutusi (vt punktis 2.3.2 kirjeldatud lahendust, lk 18). Selleks kasutatakse tabelit, mis sisaldab ajahetke ja bloobi, kuhu on kodeeritud sel hetkel muutunud signaalide väärtused. Et igal signaalil pole oma veergu, ei ole ka vajadust igal real korrata mitte muutunud väärtusi.

Lisaks on andmebaasis ka kokkuvõttetabelid, mis sisaldavad signaalide miinimumi, maksimumi ja viimast väärtust teatud perioodil (vt punktis 2.4.2 kirjeldatud lahendust, lk 21). Kokkuvõttetabelitest tehakse omakorda kokkuvõtteid. Eesmärgiks on lugemise kiirendamine – ülevaatliku graafiku kuvamiseks tuleb sel juhul terve andmebaasi asemel läbi skaneerida palju väiksem kokkuvõttetabel ning tänu miinimumi ja maksimumi salvestamisele ei lähe seejuures ekstreemumid kaduma.

3.2 Kasutatavad tehnoloogiad

Kasutatakse andmebaasisüsteemi SQLite.

- Kuna see on kasutusel olemasolevas süsteemis, siis on juba olemas:
    - C++ pakend SQLite C API kasutamise lihtsustamiseks;
    - abiklassid andmebaasi ja ettevalmistatud SQL lausete (*prepared statement*) loomiseks;
    - server, mis võimaldab vajadusel lugeda SQLite andmebaasi ka üle võrgu.
- Piisav andmete lisamise kiirus. Testide järgi näiteks kiirem kui PostgreSQL (vt punktides 2.3.1 ja 2.3.2 tehtud teste).

- Andmete lugemise kiirus sarnane teiste süsteemidega lihtsate päringute puhul ning realiseeritava süsteemi puhul ei vaja ükski päring tabelite ühendamist ega muid keerulisemaid asju, mis vajaks head päringuplaani koostajat [7], mis aeg-ajalt analüüsiks ridade arvu erinevates tabelites (SQLite seda ei tee [8]).

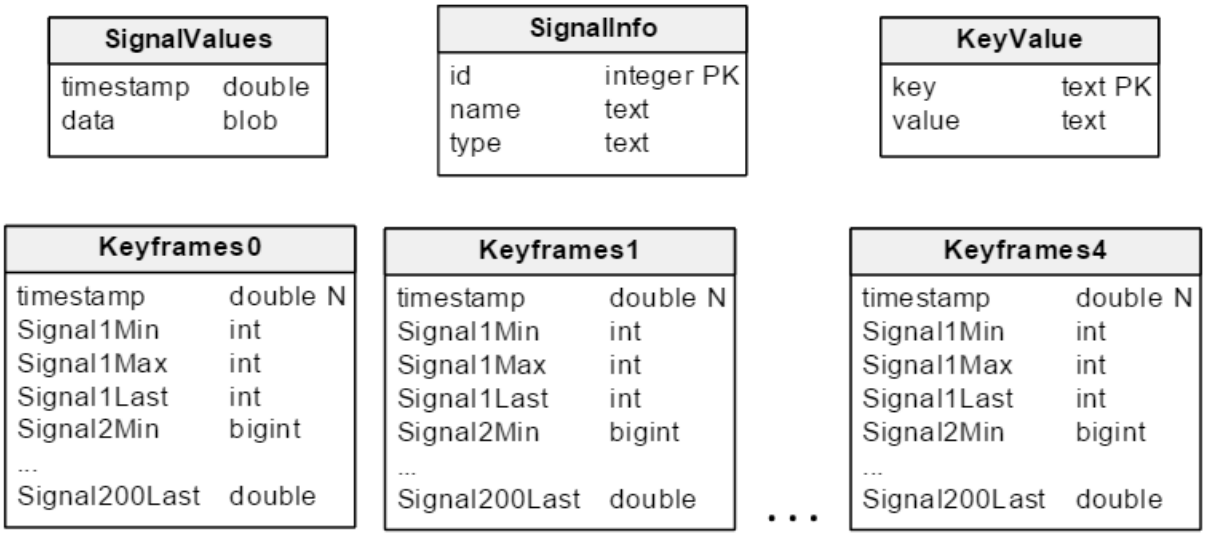
- SQLite'il pole nii head paralleeltöö tuge, kui suurematel andmebaasisüsteemidel, näiteks iga kirjutamine lukustab terve andmebaasi, mitte ainult vajalikke ridu või tabeleid [9], [10]. Logimise süsteemis on aga ainult üks kirjutaja, nii et see probleemiks ei osutu ning SQLite siiski toetab režiimi, mille korral andmebaasist saab lugeda samal ajal, kui keegi teine sinna kirjutab [4].

- Ei vaja eraldi protsessis töötavat andmebaasiserverit [11], andmebaasisüsteem on osa logimise programmist – paigaldamisel vähem konfigureerimist ja pole ohtu, et serveriga ühenduse katkemise korral tekiks andmete logimises tõrkeid.

Kokkuvõtlikult võib öelda, et kuna ei leidu häid põhjuseid SQLite'i välja vahetamiseks, siis pole põhjust seda teha. Oluline on ka see, et praegust süsteemi kasutanud insenerid on harjunud, et nad ei pea lisaks logimise rakendusele üles seadma ka eraldi andmebaasi serverit.

Kui aga kunagi peaks olema vaja andmebaasisüsteemi vahetada, siis kuna SQL keel on lihtsate päringute korral eri süsteemide puhul suhteliselt sarnane, ei ole vaja palju muuta. Selleks, et vahetus lihtsam oleks, käivad kõik pöördumised andmebaasi poole läbi CDP rakendusraamistikuga kaasa tuleva CDP2SQL liidese, millele on võimalik vajadusel lisada ka teiste andmebaasisüsteemide tuge [12].

3.3 Andmebaasi struktuur



Joonis 10 Andmebaasi struktuur.

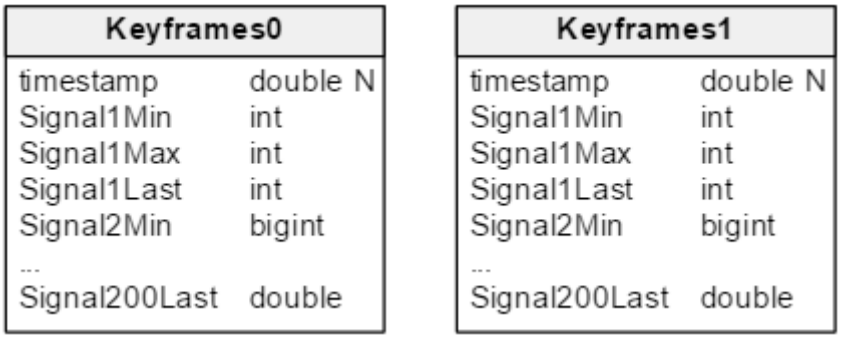
Andmebaasis on nelja erinevat tüüpi tabeleid:

- 5 kokkuvõttetabelit – sisaldavad signaaliväärtuste miinimumi, maksimumi ja viimast väärtust teatud perioodil. Eri tabelites vastab üks rida eri pikkusega perioodile.
- Signaalimuutuste tabel – sisaldab iga signaali igat muutust.
- Signaaliinfo tabel – säilitab vastavust signaali ID, nime ja tüübi vahel.
- Võtme-väärtuse tabel – sinna salvestatakse süsteemi tööks vajalikke parameetreid.

3.3.1 Kokkuvõttetabelid

Igas kokkuvõttetabelis on üks veerg ajahetke jaoks („timestamp“) ning 3 veergu iga logitava signaali kohta: minimaalne, maksimaalne ja viimane väärtus. Erinevates kokkuvõttetabelites tähistab üks rida erineva pikkusega perioodi. Ajahetke veerg on indekseeritud, sest kõiki päringuid hakatakse selle järgi tegema.

Ajahetke veerg on tüüpi *double*, ülejäänud veerud luuakse vastavalt logitava signaali tüübile (*tinyint*, *smallint*, *int*, *bigint*, *float* või *double*).



Joonis 11 Kokkuvõttetabelite struktuur.

Tabel 6 Näide kokkuvõttetabeli sisust.

	Timestamp	Signal1Min	Signal1Max	Signal1Last
1	2,06	0	205	100
2	4,14	15	250	200

Väärtus ajahetkeveerus tähistab ajaperioodi lõppu. Näiteks ülalpool oleva tabeli teine rida tähistab signaali väärtusi ajaperioodil 2,06–4,14.

Tabel 7 Esimene kokkuvõttetabel, mille põhjal tehakse teine.

	Timestamp	Signal1Min	Signal1Max	Signal1Last
1	0	0	5	3
2	1	2	10	5
3	2	4	15	7
4	3	6	20	9
5	4	8	25	11
6	5	10	20	13
7	6	12	20	15

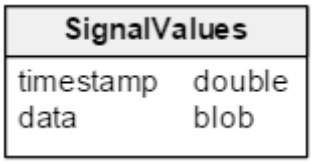
Tabel 8 Teine kokkuvõttetabel, tehtud esimese põhjal

	Timestamp	Signal1Min	Signal1Max	Signal1Last
1	6	0	25	15

Kokkuvõttetabelitest tehakse omakorda kokkuvõtteid. Näiteks teise kokkuvõttetabeli (Tabel 7) esimene rida koosneb esimese kokkuvõttetabeli (Tabel 8) esimest 7 reast.

3.3.2 Signaalimuutuste tabel

Sisaldab ajahetke ja sellele vastavat bloobi, kuhu on kodeeritud eelmise reaga võrreldes muutunud signaalide väärtused. Sarnaselt kokkuvõttetabelitele on ajahetke veerg indekseeritud.



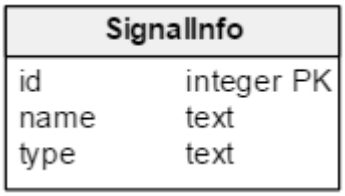
Joonis 12 Signaalimuutuste tabel.

Bloobi sisu:

- Esimesed kaks baiti tähistavad bloobi pikkust.
- Järgneb nimekiri signaali ID ja väärtuse paaridest. Iga ID võtab kaks baiti, väärtus vastavalt andmetüübi suurusele, näiteks *char* ühe, *double* kaheksa baiti. Signaali ID, nime ja tüübi vastavus on kirjas Signaaliinfo tabelis (vt punkt 3.3.3, lk 28).

3.3.3 Signaaliinfo tabel

Selles tabelis hoitakse vastavusi signaali ID, nime ja andmetüübi vahel.

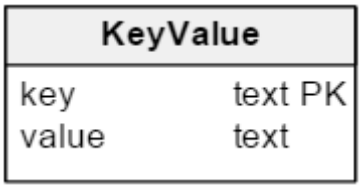


Joonis 13 Signaaliinfo tabel.

ID-d ja andmetüübid on vajalikud, et dekodeerida signaalimuutuste tabelisse salvestatud bloobi.

3.3.4 Võtme-väärtuse tabel

Sisaldab võtme-väärtuse paare ehk erinevaid parameetreid, mida on vaja süsteemi tööks salvestada.



Joonis 14 Võtme-väärtuse tabel.

Nendeks parameetriteks on:

1. Esimese kokkuvõttetabeli rea perioodi pikkus.
2. Tegur, mis näitab, mitu korda on igas järgmises kokkuvõttetabelis rea periood pikem.
3. Kokkuvõttetabelite arv.
4. Andmebaasi skeemi versioon.

Ainus neist, mis sõltub kasutaja valikutest, on esimese kokkuvõttetabeli rea perioodi pikkus, mis arvutatakse välja logimise sageduse järgi.

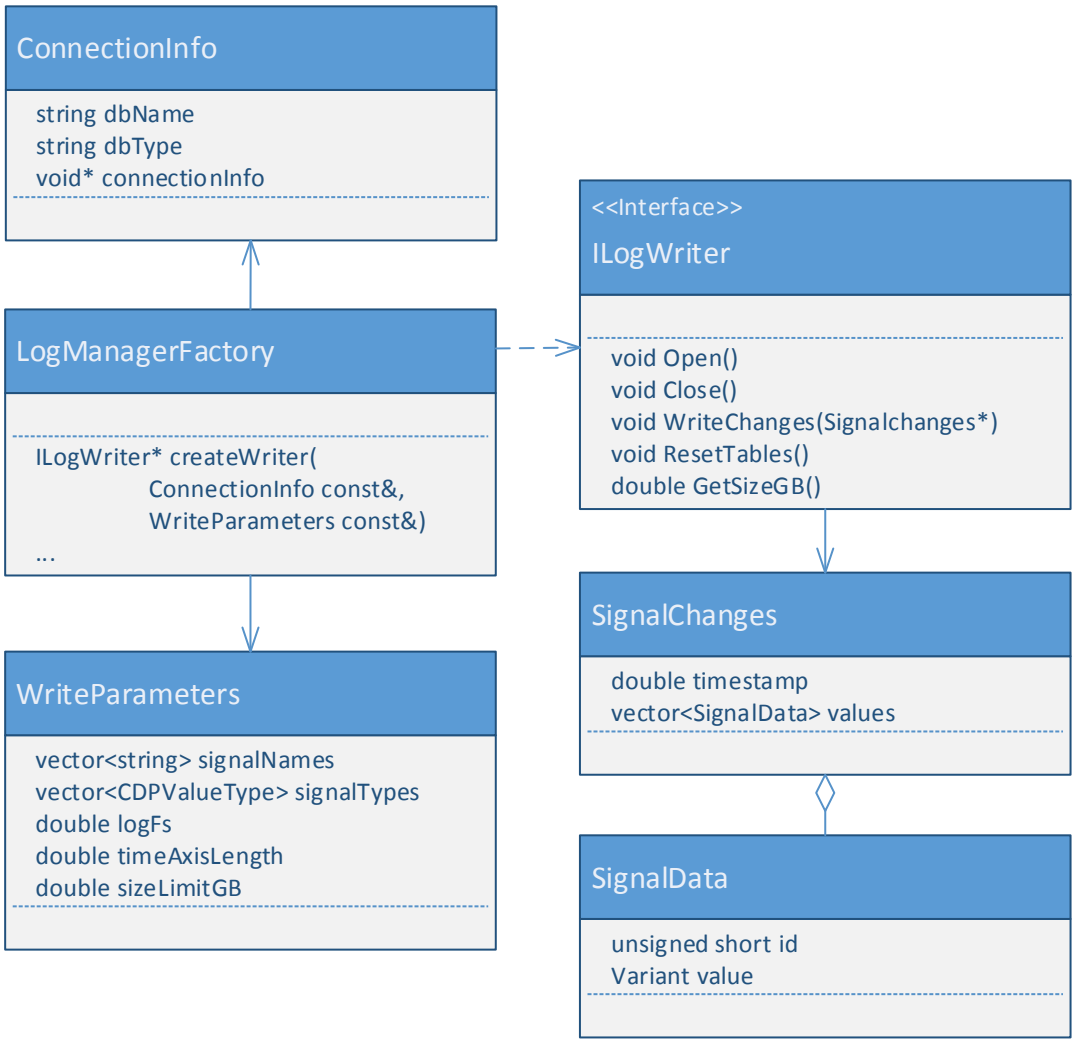
Teine ja kolmas parameeter võiksid vabalt olla konstandid koodis, sest kasutaja neid muuta ei saa. Siiski otsustasin need andmebaasi salvestada, sest see tähendab, et nende konstantide väärtusi oleks võimalik tulevikus muuta nii, et vanadest andmebaasidest saaks edasi lugeda.

Muidu tekiks probleem, millise kokkuvõttetabeli poole lugemisel pöörduda, et saada vajalik andmehulk.

Et lihtsustada tulevikus tehtavaid muudatusi, on lisatud ka andmebaasi skeemi versioon. Nii on kergem hoida ühilduvust vanade andmebaasidega.

3.4 Andmete logimine

3.4.1 API kirjeldus



Joonis 15 Kirjutamise API.

Andmete logimiseks kasutatakse liidest „ILogWriter“, mille tähtsamad meetodid on kirjeldatud allpool.

Tabel 9 Liidese „ILogWriter“ tähtsamad meetodid.

Tulemi tüüp	Nimi	Argumendid	Kirjeldus
void	Open	-	Avab ühenduse andmebaasiga, selle puudumisel loob selle koos vajalike tabelitega.
void	WriteChanges	SignalChanges* – viit struktuurile, mis sisaldab signaalimuudatusi.	Kirjutab argumendiks antud struktuuri sisu andmebaasi. Vajadusel loob ridu kokkuvõttetabelite jaoks.
void	Close	-	Loob iga kokkuvõttetabeli jaoks rea, et märkida signaali väärtused kuni sulgemishetkeni. Seejärel sulgeb ühenduse andmebaasiga.

Andmebaasi ühe rea lisamiseks tuleb kutsuda meetodit „WriteChanges“, millele antakse ette andmestruktuur „SignalChanges“, mis sisaldab ajahetke ja muutunud signaalide väärtusi (vt täpsemalt Joonis 15, lk 29).

Et „ILogWriter“ on liides, siis seda implementeeriva objekti eksemplari loomiseks kasutatakse klassi „LogManagerFactory“. Meetodi argumentideks on ühenduse loomiseks vajalik „ConnectionInfo“ ja muude logimise parameetritega andmestruktuur „WriteParameters“ (vt täpsemalt Joonis 15, lk 29).

Tabel 10 Klassi „LogManagerFactory“ tähtsamad meetodid.

Tulemi tüüp	Nimi	Argumendid	Kirjeldus
ILogWriter*	createWriter	• ConnectionInfo – ühenduse loomiseks vajalikud parameetrid • WriteParameters – muud logimise parameetrid	Loob „ILogWriter“ liidest implementeeriva objekti.

Sellise lahenduse eelis on see, et kliendi kood ei sõltu ühest kindlast implementatsioonist ja vajadusel on kerge asendada seda teistsuguse logiandmete salvestamise mehhanismiga. Eriti kasulik on see testide kirjutamisel, sest päris andmebaasi loomine iga testi jaoks oleks aeganõudev.

3.4.2 Andmete kirjutamine

Meetodiga „WriteChanges“ antakse ette signaalimuutused, mis kodeeritakse bloobiks punktis 3.3.2 (vt lk 27) kirjeldatud kujul ja kirjutatakse andmebaasi. Esialgne signaalimuutuste struktuur aga salvestatakse massiivi.

Kui on möödunud piisav aeg, luuakse massiivis olevate signaalimuutuste ja viimase kokkuvõttetabeli rea alusel uus rida kokkuvõttetabeli jaoks. Viimane kokkuvõttetabeli rida on vajalik selleks, et leida algväärtus mitte muutunud signaalidele. Selleks kasutatakse viimase väärtuse veergu (vt tabeli struktuuri lk 26, Joonis 8 Kokkuvõttetabelite struktuur).

Sarnaselt signaalimuutustest kokkuvõtete tegemisele, tehakse piisava aja möödudes kokkuvõtetest omakorda kokkuvõtteid.

See, kui tihti teha kokkuvõtteid, arvutatakse välja struktuuriga „WriteParameters“ (vt täpsemalt Joonis 15, lk 29) kaasa antud muutuja „logFs“ abil. Viimane ütleb, kui tihti logimise süsteem kontrollib, kas signaalide väärtused on muutunud. Esimesse kokkuvõttetabelisse tekib ridu 20 korda harvemini, igasse järgmisse 10 korda harvemini kui eelmisesse. Mainitud konstandid on valitud kompromissina kiiruse ja kasutatava andmemahu vahel.

Et andmete lugemisel oleks teada, millise tabeli poolde pöörduda, salvestatakse võtme-väärtuse tabelisse vajalikud parameetrid:

- Esimese kokkuvõttetabeli rea perioodi pikkus.
- Tegur, mis näitab, mitu korda on igas järgmises kokkuvõttetabelis rea periood pikem.

3.4.3 Kirjutuskiiruse tõstmine

SQLite puhul on kiiruse mõttes väga oluline kasutada suuri transaktsioone. Kui lisada ridu nii, et iga lisamine on omas transaktsioonis, siis on kiirus alla 100 rea sekundis, suurte transaktsioonide korral võib kiirus olla vabalt ka üle 20 000 rea sekundis [13]. Põhjuseks on see, et iga transaktsiooni alguses andmebaasi fail avatakse ja lõpus suletakse [14].

Seega, selleks et logimine oleks kiire, tuleks teha transaktsioonid võimalikult suureks. Sellel on aga kaks puudust:

- Mida pikem transaktsioon, seda rohkem infot läheb kaduma veaolukorra tekkimisel.

- WAL-fail kirjutatakse põhiandmebaasi faili transaktsiooni lõpus [4]. Pika transaktsiooni korral see aga kasvab väga suureks ja teeb andmebaasist lugemise aeglasemaks [4].

Et logimise sagedus sõltub kasutaja valikutest, siis otsustasin, et parem on piirata transaktsiooni suurust aja, mitte ridade arvu järgi. Kui viimasest meetodi „WriteChanges“ välja kutsumisest on möödunud rohkem kui sekund, siis kehtestatakse käimasolev transaktsioon ning alustatakse uut.

3.4.4 Andmebaasi suuruse piiramine

Olemasolev lahenduse korral sai andmebaasi maksimaalset suurust määrata ridade arvuga. Kui limiit ületati, hakati UPDATE lausete abil algust üle kirjutama. Selline meetod valiti, sest see oli testide järgi kiireim: limiidi ületamise järel logimise kiirus ei muutunud.

Kuna iga rida oli täpselt sama pikk (sisaldas iga signaali väärtust), siis andmebaasi poolt hõivatud kõvaketta ruum ei muutunud. Seega – ehkki kasutajale oli ebamugav ridade arv gigabaitideks teisendada – sai ta olla kindel, et andmebaas ei paisu kunagi nii suureks, et ruum otsa saaks. Uue logimise süsteemi puhul aga logitakse ainult signaalide muutusi, mida on vahel rohkem, vahel vähem. Järelikult on read erineva pikkusega, seega ridade arvu gigabaitideks teisendada ei saa.

Teine võimalus oleks suurust piirata nii, et limiiti ületades kustutatakse ära hulk vanemad kirjeid.

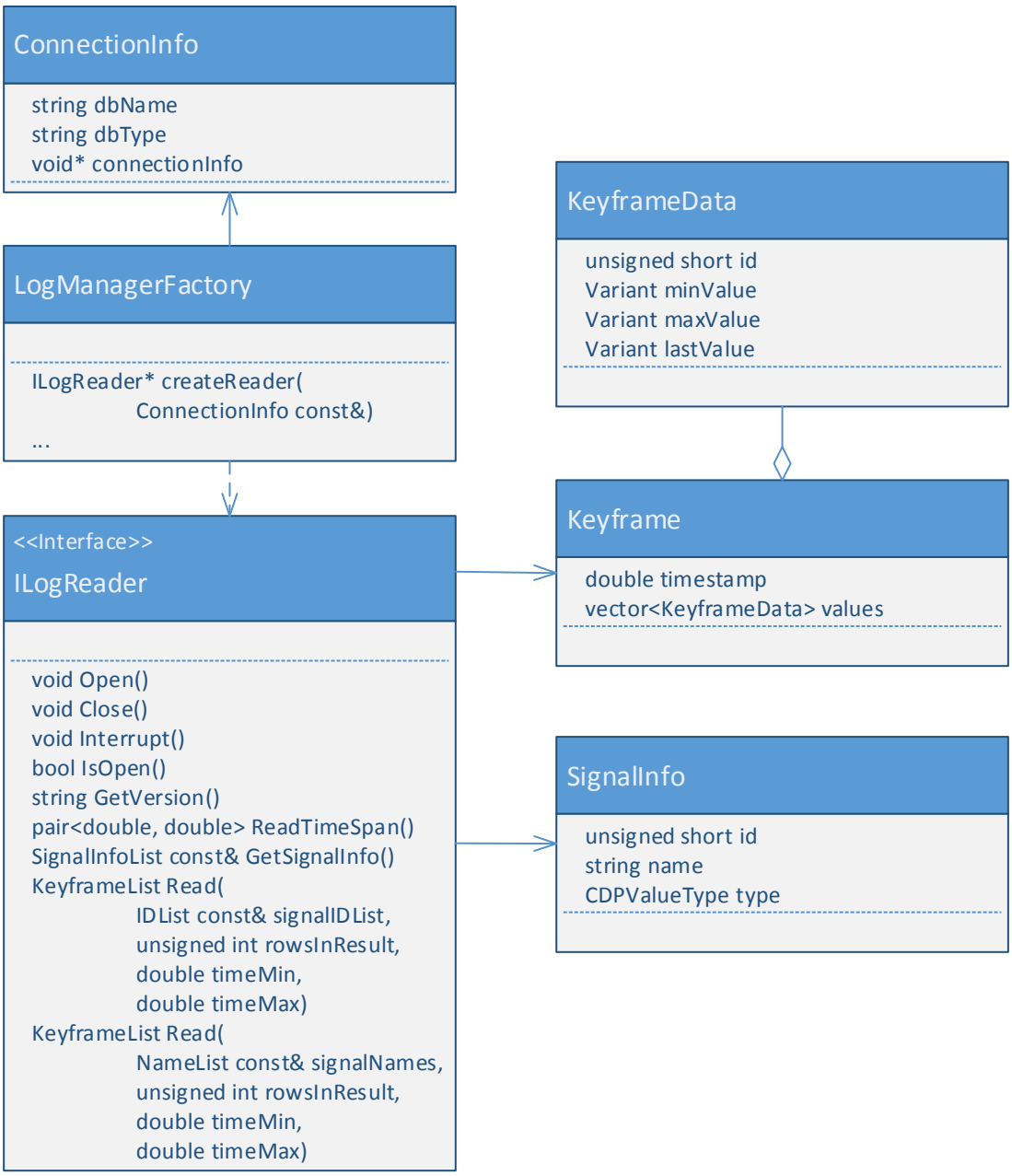
Tabel 11 Suuruse piiramise lahenduste võrdlus.

	UPDATE laused	Vanemate kirjete kustutamine
Eelised	• Maksimaalne logimise kiirus ei lange limiidi ületamisel.	• Piirab andmebaasi suurust ka siis, kui read pole sama pikad. • Limiidi ühikuks saab kasutada päevade arvu või gigabaite.
Puudused	• Andmebaasi suurus püsib konstantsena ainult siis, kui iga rida on sama suur (ei sisalda näiteks muutuva suurusega bloobe). • Kasutajale on ridade arv ebamugav ühik.	• Logimise kiirus langeb tehtud testi järgi umbes 15% limiidi ületamisel. • Kiiruse mõttes tuleks korraga kustutada hulk kirjeid.

Kaaludes eeltoodud eeliseid ja puudusi, otsustasin, et vanemate kirjete kustutamine on mõistlikum valik, eelkõige lähtudes sellest, et UPDATE lausete korral on kasutajal väga ebamugav määrata ridade arvu, teades näiteks, et andmekandjal on vaba ruumi 30 GB. Arvu määramise teeks uue süsteemi puhul eriti keeruliseks veel see, et read sisaldavad bloobe muutunud signaalidest ja võivad seega olla eri suurusega.

3.5 Andmete lugemine

3.5.1 API kirjeldus



Joonis 16 Lugemise API.

Andmete lugemiseks kasutatakse liidest „ILogReader“, mille tähtsamad meetodid on toodud allpool.

Tabel 12 Liidese „ILogReader“ tähtsamad meetodid.

Tulemi tüüp	Nimi	Argumendid	Kirjeldus
void	Open	-	Avab ühenduse andmebaasiga ja loeb mällu võtme-väärtuse tabelist tööks vajalikud parameetrid ning signaaliinfo tabelist signaalide nimekirja.
SignalInfoList	GetSignalInfo	-	Tagastab nimekirja signaalide nimedest ja ID-dest.
std::pair <double, double>	ReadTimeSpan	-	Tagastab ajatelje minimaalse ja maksimaalse väärtuse. Ehk siis perioodi logimise algusest viimase sissekandeni.
KeyframeList	Read	• IDList – nimekiri signaalide ID-dest, mille kohta soovitakse infot. • unsigned int rowsInResult – maksimaalne ridade arv tulemuses. • double timeMin – algajahetk. • double timeMax – maksimaalne lõppajahetk.	Pärib andmebaasist küsitud ID-dega signaalide väärtused algajahetkest lõpphetkeni (mõlemad otsad sisse arvatud). Tulemuseks on andmestruktuur, mis sarnaneb kokkuvõttetabeli reale ja sisaldab endas signaali miinimumi, maksimumi ja viimast väärtust igal lõigul, mille pikkuse määrab teine parameeter: maksimaalne ridade arv tulemuses.
KeyframeList	Read	• signalNames • rowsInResult • timeMin • timeMax	Sama, mis ülemine, ainult et pärida saab signaalide nimede, mitte ID-de järgi.
void	Interrupt	-	Seda meetodit võib välja kutsuda teisest lõimest, et katkestada töös olev päring. Kasulik näiteks kasutajaliideses päringu tühistamise nupu realiseerimiseks.

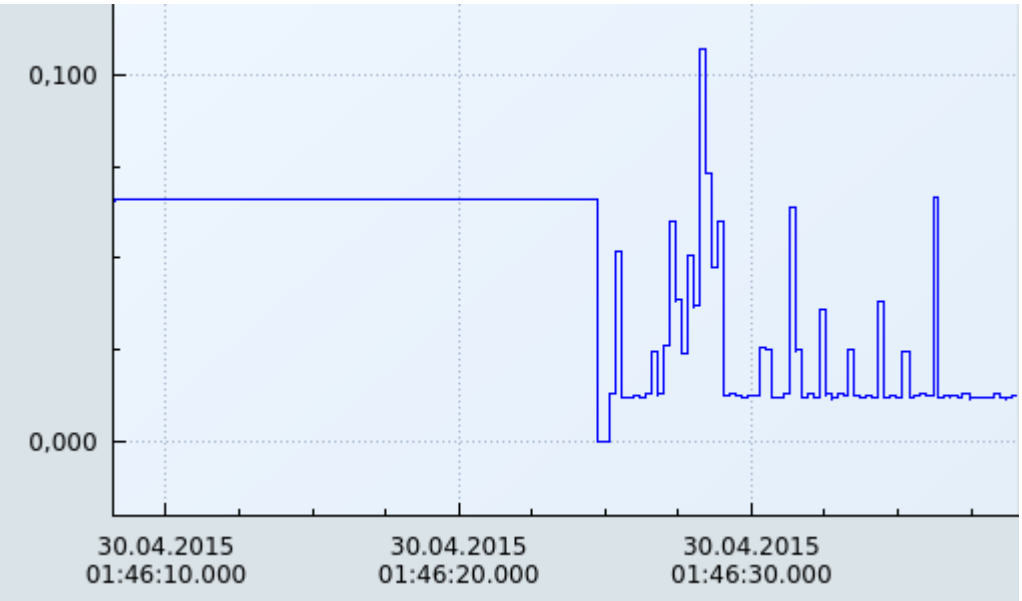
Pärast andmebaasi avamist peaks kasutaja esmalt lugema meetodiga „GetSignalInfo“ andmebaasis olevate signaalide nimed ja ID-d, sest päringu tulemused meetodist „Read“ sisaldavad väärtuste kõrval ID-si, mille järgi tunneb ära, millise signaali juurde väärtus kuulub.

Meetod „Read“ tagastab nimekirja andmestruktuuridest (vt „Keyframe“ lk 34, Joonis 16), mis sisaldavad igaüks järgmist:

- Ajahetk. Sarnaselt punktis 3.3.1 kirjeldatud (vt lk 26) kokkuvõttetabeli ridadele tähistab see ajahetk perioodi lõppu. Alguseks on eelmise rea ajahetk.
- Nimekiri signaalide väärtusi sisaldavatest struktuuridest, mille sisu on järgmine:
    - Signaali ID
    - Miinimum
    - Maksimum
    - Viimane väärtus

Meetod „Read“ garanteerib järgmist:

- Ei tagastata rohkem ridu, kui päringuga kaasa antud maksimaalse ridade arvu parameeter ütleb. Vajadusel liidetakse mitu andmebaasist loetud rida kokku üheks pikemaks perioodiks. Ei ole mõtet tagastada rohkem ridu, kui kasutaja ekraanil graafiku jaoks piksleid on.
- Kui päringusse jääb lõik, kus ei muutunud ükski signaal, siis selle lõigu kohta andmebaasi midagi ei kirjutatud ja seega ridu ei tagastata. Graafiku joone saab tõmmata selle koha peale eelmise rea viimase väärtuse abil (vt Joonis 9, lk 22).
- Iga päringu korral kaasatakse ka tulemusse esimene rida, mis on väiksem või võrdne päringu algajahetkega. Nii saab näidata graafiku vasakus servas signaali algväärtust. Tagastatud rea ajahetkeks pannakse päringu miinimumajahetk.
- Kui päritud ajavahemikus ei ole andmebaasis ühtegi rida, siis tagastataksegi ainult eelpool mainitud algväärtus, sest rea puudumine tähendab, et signaal pole muutunud.



Joonis 17 Päring tagastab alati signaali algväärtuse.

Sarnaselt punktis 3.4.1 (vt lk 29) kirjeldatud liidese „ILogWriter“ loomisele, luuakse ka „ILogReader“ klassi „LogManagerFactory“ abil.

Tabel 13 Klassi „LogManagerFactory“ tähtsamad meetodid.

Tulemi tüüp	Nimi	Argumendid	Kirjeldus
ILogReader*	createReader	• ConnectionInfo – ühenduse loomiseks vajalikud parameetrid	Loob „ILogReader“ liidest implementeeriva objekti.

3.5.2 Õige tabeli poole pöördumine

Vahekokkuvõtete loomise mõte oli see, et andmete lugemisel saaks pöörduda ainult õige resolutsiooniga kokkuvõttetabeli poole ja niiviisi täita päring kiiremini. Otsustamaks, milline tabel on kõige õigem, kasutatakse järgnevaid päringuga kaasa antud parameetreid:

- Algajahetk.
- Lõppajahetk.
- Oodatav ridade arv. See põhimõttelisest näitab, mitu pikslit lai on graafiku aken kasutaja ekraanil.

Nende kolme abil saab välja arvutada, kui pikka ajavahemikku üks piksel täidab. Seda väärtust tuleb seejärel võrrelda sellega, kui pikka ajavahemikku tähistab üks rida mingis

kokkuvõttetabelis. Viimase arvutamiseks on vaja võtme-väärtuse tabelisse salvestatud parameetreid:

- Esimese kokkuvõttetabeli rea perioodi pikkus.
- Tegur, mis näitab, mitu korda on igas järgmises kokkuvõttetabelis rea periood pikem.

Nende andmete põhjal pöördutakse esimese tabeli poole, milles on ridu tihedamalt, kui vaja ekraanil iga piksli juurde ühe väärtuse andmiseks.

3.5.3 Kokkuvõttetabelist lugemine

Kui on pöördutud õige kokkuvõttetabeli poole, ei tagastata päringu tegijale siiski kõiki ridu, vaid ainult nii palju, kui küsiti, või vähem, kui päringu perioodi jääb lõik, kus andmeid ei logitud. Selle jaoks tuleb read jagada gruppideks ning kokku liita ja selleks kasutatakse sarnaselt olemasolevale süsteemile (vt lk 15) GROUP BY päringut. Üks näide oleks selline:

```
SELECT timestamp, MIN(Signal0Min), MAX(Signal0Max), Signal0Last
FROM Keyframes2
WHERE timestamp BETWEEN 1 AND 415605.81
GROUP BY (CAST(x_axis/(415604.81 / 300) AS INT)) ORDER BY MAX(timestamp);
```

Mainitud signaalide algväärtuste leidmiseks (vt Joonis 17, lk 37) tehakse eraldi päring ja lisatakse tulemusse.

3.5.4 Signaalimuutuste tabelist lugemine

Signaalimuutuste tabeli poole pöördutakse siis, kui graafikul on pilt nii palju sisse suurendatud, et kokkuvõttetabelite infost enam ei piisa. Lugemisel on sammud järgmised:

1. Loetakse esimest kokkuvõttetabelist signaalide algväärtused. Et signaalimuutuste tabelis on ainult muudatused, siis igale signaalile algväärtuse leidmiseks tuleks halvimal juhul skaneerida läbi kõik read kuni tabeli alguseni. Seega on mõistlikum võtta need kokkuvõttetabelist.
2. Loetakse read signaalimuutuste tabelist. Päring sinna tehakse alates eelmises punktis loetud kokkuvõtterea viimase väärtuse ajahetkest, mitte kasutaja päritud algajahetkest, sest nende kahe vahel võis signaal juba muutuda.
3. Eelmises punktis loetud bloobid dekodeeritakse (vt tabeli ja bloobi struktuuri punktist 3.3.2, lk 27).

4. Päringu tegijale tuleb tagastada küsitud arv ridu. Selleks tuleb võib-olla mitu rida kokku liita. Et signaalimuutuste tabelis hoiti andmeid bloobi kujul, siis ei saa seda teha SQL keele võimaluste abil nagu kokkuvõttetabelist lugemisel (vt punkt 3.5.3, lk 38), kus seda tehti GROUP BY abil, vaid read tuleb käsititsi grupeerida ja liita. Selleks:

    a. Arvutatakse välja gruppide piirid.

    b. Kui esimene grupp jäi tühjaks, siis pannakse sinna signaalide algväärtused.

    c. Liidetakse kokku kõik ühes ja samas grupis olevad read, nii et alles jääb perioodi miinimum, maksimum ja viimane väärtus.

        i. Kui gruppi sattus ainult üks rida, siis need kolm väärtust on võrdsed (juhtub, kui graafikule on lõpuni sisse suurendatud).

        ii. Kui gruppi ei sattunud ükski rida, siis päringu tulemusse selle koha peale rida ei tekitata. Graafiku joonistamisel kasutatakse eelmiselt realt võetud viimast väärtust.

Ehkki andmete formaat on signaalimuutuste tabelis testsugune kui kokkuvõttetabelites, siis need teisendatakse ja päringu tegijale erinevus välja ei paista.

4. Lahenduse testimine

Testitakse vastavust mittefunktsionaalsetele nõuetele (lk 15).

4.1 Logimise kiirus

Terviklikkus logimise süsteemis saavad CDP signaalid liikuda maksimaalselt 1000 korda sekundis [15]. Seega, et saada paremat aimu sellest, kui kiiresti selle töö käigus loodud andmete kirjutamise vahekiht toimib, on mõistlik terve süsteemi testimise asemel kutsuda otse välja meetodit „WriteChanges“.

Esimene mittefunktsionaalne nõue (lk 15) oli, et logimise kiirus peab olema vähemalt 1000 korda sekundis 200 signaali korral. Et loodud logimise süsteem võetakse kasutusele riistvaral, mis on oluliselt nõrgem kui tavaarvuti, siis vastavust sellele kontrolliti Matrix MXC-2002(G) kontrolleril [16], millel on järgnevad omadused:

- Intel® Atom™ N270 1.6 GHz protsessor (välja tulnud 2008. aastal, TDP 2.5 W)
- 1 GB RAM
- 32 GB SSD
- Puuduvad liikuvad osad, hea vastupidavus vibratsioonile.

Tabel 14 Logimiskiiruse test kontrolleril.

Ridu andmebaasis	150 000
Andmebaasi suurus	73,8 MB
Kulunud aeg	48,9 s
Ridu sekundis	3067

Nagu näha, siis logimise kiirus on üle kolme korra kiirem miinimumnõudest, seega võib eeldada, et see ei jää süsteemis pudelikaelaks.

4.2 Graafiku kuvamise kiirus

Graafiku kuvamise kiiruse testimiseks tuleb esmalt luua andmebaas. Selleks käivitasin olemasoleva ja uue logimise süsteemi järgmiste parameetritega:

- 200 signaali, neist 40 muutusid iga takt, 160 ei muutunud kunagi.
- Logiti 100 korda sekundis 9 tundi ja 23 minutit.

Seejärel sai võrrelda olemasolevat süsteemi uuega. Nagu punktis 2.1.3 (vt lk 14) mainitud, oli vana süsteemi korral lugemiseks kaks võimalust:

- Laadida enne lugemist andmebaasist kõik vajalikud signaaliväärtused mällu. See tuleb kõne alla ainult lokaalse faili puhul.
- Teha GROUP BY päring, et tagastataks ainult nii palju ridu, kui on ekraanil piksleid. Seda kasutatakse enamasti üle võrgu andmebaasi vaatamiseks.

Võrdluseks testisin mõlemat nii, et korraga kuvatakse graafikul 16 signaali 200-st. Esimene test on tehtud oma arvutis lokaalset faili avades, andmebaas asus SSD-l.

Tabel 15 Lugemiskiiruse testimine lokaalse faili korral 16 signaaliga.

	Olemasolev süsteem		Uus süsteem
Andmebaasi eellaadimine mällu.	Jah	Ei	Ei
Andmebaasi suurus	7,1 GB		1,7 GB
Esialgse graafiku kuvamine	69,4 s	79,5 s	0,11 s
Suurendamine	kuni 5 s	kuni 67 s	kuni 0,52 s

Nagu näha, siis uue süsteemi puhul on esialgse graafiku kuvamine oluliselt kiirem. Ka ei vaja see kõikide signaalimuutuste mällu laadimist selleks, et suurendamine kiire oleks. Tulemused näitavad, et uus süsteem vastab teisele mittefunktsionaalsele nõudele (lk 15), mille järgi graafiku avamine ega suurendamine ei tohi võtta üle kahe sekundi.

Ehkki läbi serveri andmete kuvamise kohta otsest kiirusenõuet polnud, otsustasin siiski ka seda testida, sest siis saab andmebaasi faili hoida kontrolleri peal ja näeb jõudlust nõrgemal riistvaral. See test näitab andmete lugemise kiirust lokaalvõrgus. Kui ühendus luuakse maismaalt laevaga läbi satelliidi, võivad tulemused suure hilistumise ja aeglase kiiruse tõttu kehvemad olla.

Tabel 16 Lugemiskiiruse test läbi kontrolleril asuva serveri 16 signaaliga.

	Olemasolev süsteem	Uus süsteem
Andmebaasi eellaadimine mällu.	Ei	Ei
Andmebaasi suurus	7,1 GB	1,7 GB
Esialgse graafiku kuvamine	430,9 s	1,67 s
Suurendamine	kuni 417 s	kuni 5,2 s, tavaliselt 1,5–2,5 s

Nagu näha, siis vana süsteemi kasutamine läbi serveri on suhteliselt vaevaline. Uus süsteem töötab aga isegi 16 signaaliga talutava kiirusega. Enamus päringuid võtab sel juhul aega 1,5–2,5 sekundit, kuigi mõned ka kuni 5,2 sekundit.

Selgituseks tuleks mainida, et vana süsteemi aegluse tõttu kasutajad tihti ei soovinud alustada kogu määramispiirkonna ülevaatest, mida siin testiti, vaid sisestasid kohe käsitsi lühikese ajavahemiku, et vaadata ainult lõiku, kus kõige tõenäolisemalt toimus neid huvitav sündmus. Kuna sel juhul oli töödeldavate andmete hulk väiksem, täideti päring kiiremini. Uue süsteemi puhul pole selline tegevus aga enam vajalik. Võimalik, et tänu täieliku ülevaate nägemisele märkavad kasutajad nüüd ka midagi, mis oleks muidu kahe silma vahele jäänud.

5. Võimalikud edasiarendused

5.1 Logimise lõpu märkimine

Praeguse lahenduse korral eeldatakse, et ridade puudumine andmebaasis tähendab, et ükski signaal ei muutunud, mitte et logimine lõppes. Seega, kui andmete logija oleks mingil perioodil olnud välja lülitatud, siis oleks graafikul sirge joon, mis näitab, et andmed ei muutunud – see aga ei pruugi tõsi olla.

Lihtsaim võimalus oleks ilmselt sulgemisel kirjutada lõpu tähistamiseks andmebaasi rida väärtustega NULL, aga siis tuleks muuta lugemise osa, et see sellega arvestaks. Lisaks jääks lahendamata probleem, mis saab siis, kui näiteks logiv kontroller voolu kaotab ning seega rida väärtustega NULL kirjutamata jääb. Lahendusi on mitmeid ja see vajaks lisaanalüüsi.

5.2 Kokkuvõtvate andmete saatmine maismaaserverisse

Kuna ühendus läbi satelliidi laevadega on väga aeglane ja kõikuv, võiks andmetest olla koopia maismaal. Ülesandeks oleks luua süsteem, mis loob turvalise ühenduse maismaaserveriga ja saadab sinna aeg-ajalt uued logid. Ühendus on aga piisavalt aeglane (kuni 250 kbit/s), et ei saaks saata kõiki signaalimuutusi, vaid ainult vahekokkuvõtteid.

Serveri poolel tuleks analüüsida, kas on mõtet jätkata selle töö käigus kontrollerite jaoks loodud C++ ja SQLite kooslusega või vahetada need mõne muu tehnoloogia vastu. Arvestada tuleks kasutatava platvormi eripärade, koormuse, kasutusjuhtude ning ka sellega, et serveril ilmselt ei ole andmemahu kasutus nii suureks probleemiks kui kontrolleril ja ehk oleks mõistlik seal keerukust lisavaid bloobe mitte kasutada.

5.3 Detailsemate andmete jupiti pärimine

Eelmises punktis kirjeldatud maismaaserveris saaks olla ainult ülevaatlikud kokkuvõtted andmetest. Samas oleks vahel, näiteks veaolukorra uurimiseks, vaja maismaal asuval meeskonnal uurida ka detailsemaid logiandmeid.

Sel juhul võiks maismaaserver pärida laevast vajalikud lõigud ja salvestada need ka tulevaseks kasutamiseks. Viimase realiseerimise teeb keerukaks see, et on raske otsustada, millise tabeli poole lugemisel pöörduda, kui detailsemates tabelites on andmeid ainult jupiti.

6. Kokkuvõte

Antud töö eesmärgiks oli optimeerida andmete logimise ja presenteerimise süsteemi nii, et see kasutaks vähem andmemahtu ja kuvaks ülevaatliku graafiku andmetest oluliselt kiiremini kui olemasolev süsteem.

Töö käigus realiseeriti logiandmete vastuvõtja ja andmebaasisüsteemi vahele vahekiht, mis töötleb andmeid nii, et:

- Andmemahu säästmiseks salvestatakse ainult signaaliväärtuste muutused. Mitte muutunud väärtusi ei korrata. Lahendus, mis võimaldas suurimat logimiskiirust, oli kodeerida andmed binaarkujule ja salvestada need bloobina.

- Lugemise kiirendamiseks luuakse andmetest erinevatesse tabelitesse erineva resolutsiooniga vahekokkuvõtteid, mis sisaldavad iga signaali mingisugusel perioodil olnud väärtuste miinimumi, maksimumi ja viimast suurust.

Lugemise ajal see vahekiht valib, arvestades päringu parameetreid, millise resolutsiooniga tabeli poole on mõistlik pöörduda, ja tagastab tulemuse oluliselt kiiremini kui vana süsteem.

Summary

The aim of this work was to optimize a system of logging and presenting data, so that it would use less disk space and display an overview graph from logged data much quicker than the existing system.

As the result of this study, an intermediate layer was added between the logger and the database management system. Its purpose is to process the data so that:

- Only the changes of signal values would be stored. Values, which did not change, are not repeated. The solution for this problem that enabled the fastest logging speed was to code the data into binary form and store it as a BLOB.

- To speed up reading, several summary tables of the data with different resolutions are made during logging. These summaries contain the minimum, maximum and last value of each signal for a given period.

When the data is read, this intermediate layer will consider the parameters of the query, choose the first table with the needed resolution and return the result much faster than the existing system.

Kasutatud kirjandus

[1] „e-teatmik,“ [Võrgumaterjal]. Available: http://vallaste.ee/sona.asp?Type=UserId&otsing=5031. [Kasutatud 09 04 2015].

[2] „CDP - Control Design Platform,“ ICD Software, [Võrgumaterjal]. Available: http://www.icdsoftware.no/products/control-design-platform/more-information. [Kasutatud 13 04 2015].

[3] „Vikipeedia – Rakendusliides,“ [Võrgumaterjal]. Available: http://et.wikipedia.org/wiki/Rakendusliides. [Kasutatud 15 05 2015].

[4] „SQLite – Write-Ahead Logging,“ [Võrgumaterjal]. Available: https://www.sqlite.org/wal.html. [Kasutatud 11 05 2015].

[5] „PostgreSQL 9.1.15 Documentation – Starting the Database Server,“ [Võrgumaterjal]. Available: http://www.postgresql.org/docs/9.1/static/server-start.html. [Kasutatud 06 05 2015].

[6] „Qt Documentation – Qt SQL,“ [Võrgumaterjal]. Available: http://doc.qt.io/qt-5/qtsql-index.html. [Kasutatud 11 05 2015].

[7] A. Turner, „Nerds Central,“ [Võrgumaterjal]. Available: http://nerds-central.blogspot.com/2008/01/beware-ultra-fast-databases-re-sqlite.html. [Kasutatud 06 05 2015].

[8] „The Next Generation Query Planner,“ [Võrgumaterjal]. Available: https://www.sqlite.org/queryplanner-ng.html. [Kasutatud 06 05 2015].

[9] „SQLite – Appropriate Uses For SQLite,“ [Võrgumaterjal]. Available: https://www.sqlite.org/whentouse.html. [Kasutatud 11 05 2015].

[10] „PostgreSQL 9.4.1 Documentation – Concurrency Control,“ [Võrgumaterjal]. Available: http://www.postgresql.org/docs/9.4/static/mvcc-intro.html. [Kasutatud 11 05 2015].

[11] „About SQLite,“ [Võrgumaterjal]. Available: https://www.sqlite.org/about.html. [Kasutatud 06 05 2015].

[12] ICD Software, „CDP2SQL v3.5.0.0 – Programmer Manual,“ 2013.

[13] „Stack Overflow – Improve INSERT-per-second performance of SQLite?,“ [Võrgumaterjal]. Available: http://stackoverflow.com/questions/1711631/improve-insert-per-second-performance-of-sqlite. [Kasutatud 11 05 2015].

[14] „SQLite – Performance Tuning,“ [Võrgumaterjal]. Available: http://www.sqlite.org/cvstrac/wiki?p=PerformanceTuning. [Kasutatud 11 05 2015].

[15] ICD Software AS, „CDP System Manual V3.5.0,“ 2014.

[16] „MXC-2000 Series,“ [Võrgumaterjal]. Available: http://www.adlinktech.com/PD/marketing/Datasheet/MXC-2000Series/MXC-2000Series_Datasheet_en_1.pdf. [Kasutatud 13 05 2015].

[17] „PostgreSQL 9.4.1 Documentation – Database Page Layout,“ [Võrgumaterjal]. Available: http://www.postgresql.org/docs/9.4/static/storage-page-layout.html. [Kasutatud 09 05 2015].