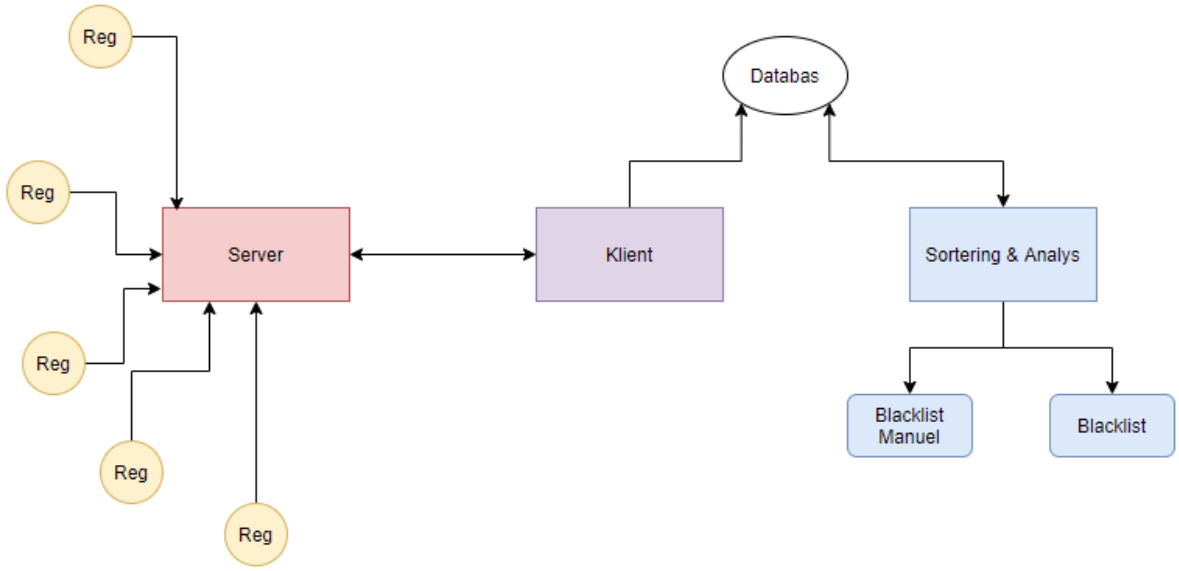
CHALMERS



Realtidssändning, hämtning och analys av regulatorer

Realtime transmission, gathering and analysis of regulators

Examensarbete inom Data- och Informationsteknik

Joachim Antfolk
Tobias Mauritzon

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2021

Realtids sändning, hämtning och analys av regulatorer

Joachim Antfolk
Tobias Mauritzon

Handledare: Sakib Sistek, Chalmers Tekniska Högskola
Examinator: Jonas Duregård, Chalmers Tekniska Högskola

Examensarbete 2021
Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola / Göteborgs Universitet
412 96 Göteborg
Telefon: 031-772 1000

Omslag:
Bilden beskriver flödet av information genom projektet med start i PID-regulatorerna och slutet i svartlistorna för felaktiga regulatorer.

Institutionen för Data- och Informationsteknik
Göteborg 2021

SAMMANFATTNING

I dagens fabriker och mindre anläggningar finns det mängder med regulatorer. Många av dessa regulatorer är inte trimmade helt rätt eller fungerar helt enkelt fel. I tidigare examensarbeten har analyshjälpmedel tagits fram för att se vilka av regulatorerna i en anläggning som inte fungerar korrekt.

Detta examensarbete bygger vidare på analysprogrammet skapat av R. Ernfjäll och J. Larsson genom att skapa funktionalitet som gör att regulatorinformation kan skickas till en server som klienter sedan kan ansluta till för att hämta data för att sedan göra en analys. Hela det beskrivna flödet har implementerats och testats i detta arbete och resultaten är lovande om antalet regulatorer per klient inte blir för många.

Mycket fokus i arbetet har lagts på att skapa server- och klientkopplingen med en OPC-koppling men den gamla koden hade problemet att analysen blev för långsam. Problemet med hastigheten har undersökts och minimerats under projektets gång. I stället för ett Excel-dokument så använder nu analysen istället en databas som kan uppdateras med ny information kontinuerligt och analysen sker på ett bestämt schema.

Projektet går också igenom skapandet samt användandet av program kompilerade från projektet och hur dessa program fungerar som separata enheter och vad de behöver för att köras.

Nyckelord: Optimering, regulator, analys, OPC, PID

ABSTRACT

In today's industrial complexes and smaller industries there are lots of regulators. Many of these regulators are not configured correctly or have stopped functioning as they should. In previous works made at Chalmers, analysis solutions have been created that look at regulators in a facility and find the faulty ones.

This project builds on the previous work done by R. Ernfjäll and J. Larsson by creating functionality that makes it possible to send information about the regulators to a server in realtime, that server can then be accessed by a client that fetches data and saves it in a database that the analysis program can reach. This process has been implemented and tested in the project and as long as there are not too many regulators the program works great.

The focus of this project has been somewhat split, the first priority was the creation of a server and client with an OPC connection but the analysis part was deemed too slow. The problem with the analysis speed was investigated and has been improved. Instead of like the old system where an Excel file was read, the new system uses a database that gets new information constantly and the analysis runs on a schedule.

This report also highlights the creation process and how to use the programs compiled from the project and how these programs function as separate entities.

FÖRORD

Ett stort tack till Veronica Olesen för att hon bidragit med sin kunskap inom området samt kommit med synpunkter under arbetet som ledde till nya synvinklar att utforska och undersöka. Vi vill också tacka vår handledare Sakib Sistek som har kommit med många synpunkter under projektet, hjälpt oss lägga fokus på rätt områden och pushat mycket för ett bra arbete och en informativ rapport.

ORDLISTA

Tillstånd	Används i denna rapport för att beteckna en regulators tillstånd i ett visst ögonblick. I detta projekt är detta en kombination av regulatorns ärvärde, börvärde, felvärde, styrsignal och läge.
Skrivbordsmiljö	En uppsättning programvara med ett sammanhängande grafiskt användargränssnitt.
Interoperabilitet	En förmåga inom olika system som låter dem arbeta och kommunicera tillsammans.
Server	En server är någon form av dator som tillhandahåller data till andra datorer.
Klient	En klient är någon form av dator som använder en tjänst som görs tillgänglig av en server.
PLC	Förkortning av *"Programmable Logic Controller".* En PLC är robust dator som används för styrning av produktionsförlopp, exempelvis ett löpande band. En PLC kan hantera både digitala och analoga signaler för både in- och utdata.
Programflöde	Den ordning som ett program utför instruktioner.
API	Förkortning av *"Application Programming Interface"*, på svenska Applikationsprogrammeringsgränssnitt. Ett API är ett gränssnitt som tillåter ett program att använda funktioner i ett bibliotek.
GUI	Förkortning av *"Graphical User Interface"*, på svenska grafiskt användargränssnitt. Ett GUI är ett gränssnitt för interaktion med ett program via grafiska komponenter, exempelvis en knapp.
CLI	Förkortning av *"Command-Line Interface"*, på svenska kommandotolk. Ett CLI är ett program för interaktion med ett program via att användaren skriver kommandon i en kommandotolk och som även kan visa resultatet av kommandot.
Kompilera	Att förvandla skriven programkod till ett format som en maskin kan förstå.
IDE	Förkortning av *"Integrated Development Environment"*, på svenska Integrerad utvecklingsmiljö. En IDE är ett program ämnat för skapande, redigering, kompilering och testning av programkod.
Tupel	I PostgreSQL är en tupel en objektssekvens där varje objekt har en bestämd typ, till exempel en text eller ett heltal.
Csv	En .csv-fil är en fil som innehåller rader av kommaseparerade värden.
Typ	När ordet typ används inom programmering menas att värdet är för något är en bestämd typ så som exempelvis ett heltal (int) eller en text (string).
Data	Data/ information eller uppdateringar syftar på information som kommer från regulatorerna antingen angående regulatorerna själva eller värden som de läser från omgivningen.

Innehållsförteckning

Tabellförteckning

Figurförteckning

1 Inledning

I dagens fabriker och mindre anläggningar finns det mängder med regulatorer. Många av dessa regulatorer är inte trimmade helt rätt eller fungerar helt enkelt fel. Detta examensarbete är en del i ett större arbete för att utveckla ett verktyg som kan hjälpa till att uppmärksamma och diagnostisera fel i regulatorkretsar. Arbetet ingår i ett forskningsprojekt på Chalmers institutionen för elektroteknik och är en fortsättning på två tidigare examensarbeten som försökte lösa detta problem.

Det första är *"Diagnostisering av regulatorer"* av R. Ernfjäll och J. Larsson **[1]** från 2017 som utvecklade ett verktyg för att hitta oscillationer i reglerkretsar och felaktigt arbetande regulatorer. Programmet från Ernfjäll och Larssons arbete har två delar: en del som läser in regulatordata och en del som analyserar denna data för att hitta möjliga fel. Den delen som sköter analys fungerar i helhet bra, men inläsningen av data är i jämförelse väldigt långsam. Analysen hittar reglerkretsar som beter sig underligt samt regulatorer som används i manuellt läge och skriver ut dessa i Matlabs terminal.

Det andra arbetet är *"Systemidentifiering och analys av olinjäriteter i styrdon"* av G. Rosin och P. Svensson **[2]** från 2018 som bestod av att utveckla ett verktyg för att utvärdera glapp och friktion i dåligt fungerande styrdon. Detta program var ämnat för att användas tillsammans med Ernfjäll och Larssons program.

1.1 Syfte

Den Matlabkod som finns utvecklad från det tidigare examensarbetet av Ernfjäll och Larsson fungerar för analys av regulatorer för att hitta regulatorer som presterar dåligt. Denna kod är dock inte anpassad för att köras i realtid och har en lång inläsningstid för regulatordata.

Huvudsyftet med detta arbete är att vidareutveckla det nuvarande programmet för att tillåta insamling av regulatordata i realtid samt lagring av denna data i en databas. Den nuvarande koden ska modifieras för att hämta data från denna databas och analysera den med ett konstant tidsintervall. Om möjligt ska även prestandan av den nuvarande analyskoden förbättras.

1.2 Mål

Huvudmålet med detta arbete är att skriva ett program som kan hämta regulatordata från en server en gång i sekunden via en koppling som använder standarden OPC som sedan ska lagras i en relationsbaserad databas. Programmet skrivet av Ernfjäll och Larsson ska modifieras så att data ska kunna hämtas från databasen och analyseras löpande med ett tidsintervall på en timme.

Det är även ett mål att de olika delarna av processen ska kunna köras oberoende av varandra. Exempelvis att delen som hämtar data lagrar den i en databas och att

analysen utförs av ett annat program som läser in från databasen. Detta skulle kunna vara användbart om insamling och analys ska hanteras på olika maskiner som kommunicerar med samma databas.

Till slut så är det ett mål att utforska vilket eller vilka språk utav: Matlab, C++ och Python som passar för vidareutveckling av programmet.

1.3 Frågeställningar

Mer exakt så ska detta arbete besvara följande frågor:

- Fungerar inläsning av processdata via OPC?
- Finns det optimeringsmöjlighter i den nuvarande Matlab-programkoden för beräkningar och analys?
- Kan presentation och överföring av analysresultat införas?
- Hur påverkas inläsnings- och beräkningstiden när antalet regulatorer ändras?
- Kan andra programspråk förbättra applikationen och om så vilka programspråk är lämpliga?

1.4 Avgränsningar

På grund av begränsad tid och att den tillgängliga hårdvaran endast är två datorer så finns det några avgränsningar för det som kommer att utföras under arbetet. Dessa avgränsningar är:

- Endast C++ och Python har valts som språk för att testa optimering.
- All kod kommer endast skrivas och testas på maskiner som använder Windows 10 Home edition, 64 bitars version.
- Tester har gjorts på olika datorer och kan variera.
- Inga fler analysmetoder skapas under detta projekt.
- Inga tester har skett med riktiga regulatorer. All regulatordata är simulerad.

2 Teknisk bakgrund

Detta kapitel går igenom de resurser och verktyg som används inom projektet. Kapitlet är uppdelat i tre delar: **Teknologi**, **Mjukvara** och **Hårdvara**. Delen **Teknologi** täcker grundläggande relationsbaserade databaser, OPC-standarden och återkopplade reglerkretsar. **Mjukvara** går igenom de språk, bibliotek och program som används. Till slut innehåller **Hårdvara** information om de maskiner som används under projektet för skrivande och testande av programkod.

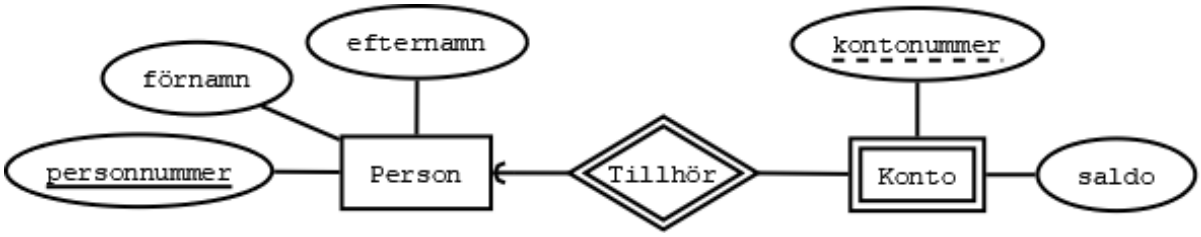
2.1 Teknologi

Detta avsnitt täcker mer generella teknologier som används i projektet. Det som behandlas är grundläggande information om relationsbaserade databaser, OPC-standarden och återkopplade reglerkretsar. För mer information om dessa ämnen se **[2]**, **[3], [4], [5], [6]** och **[7]**.

2.1.1 Relationsbaserade Databaser

En *databas* är i grunden en organiserad samling av data **[3]**. Det som används i detta projekt är en så kallad *relationsbaserad databas* som är en databas där den lagrade informationen representeras som tabeller och där varje rad har en unik identifierare som kallas en *nyckel*. Varje kolumn i en sådan tabell representerar ett attribut för den data som tabellen innehåller **[4]**. Figur 2.1 nedanför visar ett exempel för hur tabeller i en relationsbaserad databas är uppbyggda och sammankopplade i form av ett *"Entity-Relationship-schema"*, härefter benämnt ER-schema.

I Figur 2.1 finns det två *tabeller* med ett antal *attribut* var samt ett *förhållande* mellan tabellerna. Tabellen *Person* representerar en individ och har attributen *personnummer*, *förnamn* och *efternamn*, notera att personnummer är understruket vilket betyder att det är det som används som nyckel i tabellen. Den andra tabellen är *Konto* som representerar ett bankkonto som ägs av en person och har attributen *ägare*, *kontonummer* och *saldo*. Tabellen konto har en så kallad *sammansatt nyckel* vilket är en nyckel som består av flera värden, i detta fall för att representera att en person kan ha flera konton. För att se till an ägaren av kontot faktiskt existerar i *Person* så finns förhållandet *Tillhör* mellan tabellerna. Detta förhållande betyder att ett konto tillhör en person och att ett konto behöver både kontonummer och personnummer för att identifieras.



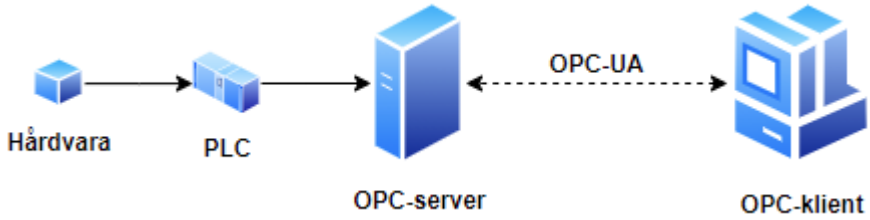
Figur 2.1: *Exempel på tabeller och relation i en relationsbaserad databas som ett ER-schema.*

2.1.2 Open Platform Communications

"Open Platform Communications", förkortat *OPC*, släpptes först 1996 och är en interoperabilitetsstandard för datakommunikation mellan industriella automatiserade system. Syftet med OPC när det släpptes var att abstrahera PLC-specifika protokoll till ett gemensamt gränssnitt för att tillåta olika system att konvertera generiska OPC-förfrågningar till enhetsspecifika förfrågningar och detsamma i motsatt riktning.**[5]**

År 2008 släpptes OPC-standarden som används i detta projekt, *"OPC Unified Architecture"* eller *OPC-UA* som är en plattformsoberoende serviceorienterad arkitektur som integrerar all funktionalitet i de tidigare OPC specifikationerna i ett nytt ramverk **[6]**. Vad som menas med att OPC-UA är en serviceorienterad arkitektur är att OPC-UA-komponenter är designade för att vara återanvändbara och interoperabla via tjänstegränssnitt **[7]**.

I figur 2.2 visas ett enkelt exempel på hur OPC-UA kan användas för kommunikation mellan OPC-server och OPC-klient. I figuren är servern kopplad till en PLC som styr hårdvara inom en process. I detta exempel så använder klienten servern för att läsa data från processen. Eftersom OPC-UA är plattformsoberoende så kan servern och klienten kommunicera även om de använder olika interna protokoll. Exempelvis så kan servern i denna koppling vara ett inbyggt system och klienten kan vara en persondator som regelbundet kontaktar servern.

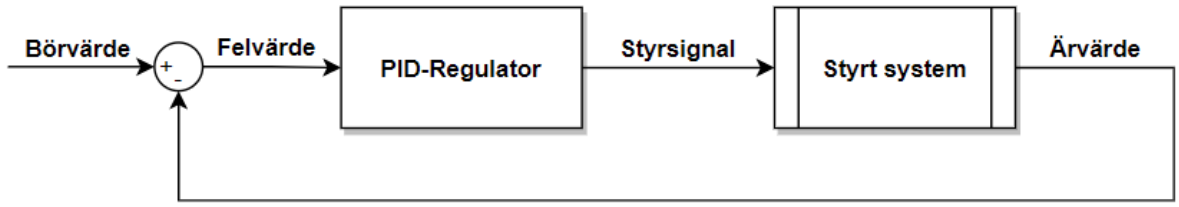


Figur 2.2: *Exempel på användning av OPC-UA för koppling mellan server och klient.*

2.1.3 Återkopplade Reglerkretsar

En återkopplad reglerkrets består generellt av en regulator, styrdon, givare och det system som ska regleras **[2]**, i detta fall så är regulatorn mer specifikt en *PID-regulator*. Figur 2.2 nedanför visar hur en återkopplad reglerkrets är uppbyggd, givare och styrdon är inkluderade i det styrda systemet i bilden. I en återkopplad reglerkrets finns det fyra signaler som brukar nämnas: *börvärde*, *ärvärde*, *felvärde* och *styrsignal* **[2]**.

Börvärdet är det tillstånd som det styrda systemet ska befinna sig i, exempelvis i ett värmeelement är den önskade temperaturen börvärdet. Ärvärdet är det styrda systemets nuvarande tillstånd. Felvärdet är differensen mellan börvärdet och ärvärdet, om differensen är noll så är systemet i rätt tillstånd och om differensen är negativ eller positiv så är systemets tillstånd för högt respektive för lågt. Styrsignalen är regulatorns utsignal och säger till det styrda systemet om det ska ändra sitt tillstånd och i så fall hur det ska ändras.



Figur 2.3: *Exempel på ett återkopplat system med en PID-regulator. Givare och styrdon är inkluderade i "Styrt system".*

En PID-regulator är en regulator vars utsignal har tre delar: en proportionell del (**P**), en integrerande del (**I**) och en deriverande del (**D**). **P**-delen är proportionell mot felvärdet, med endast denna del är det svårt att uppnå både snabbhet och stabilitet i det styrda systemet, därutöver brukar det finnas ett kvarstående fel. **I**-delen är beroende på tidigare mätvärden med **I**-delen regleras systemet utan kvarstående fel. I-delen är strängt växande om felvärdet är lika med eller större än noll och strängt avtagande om felvärdet är lika med eller mindre än noll. **D**-delen ändrar kontinuerligt på styrsignalen baserat på dess lutning, vilket gör det möjligt att få regleringen snabb och att dämpa störningar och oscillationer inom systemet.**[2]**

2.2 Mjukvara

Detta avsnitt går igenom språk, bibliotek och annan mjukvara som har använts eller testats under projektets gång. De språk som används har valts utifrån tidigare erfarenhet samt en jämförelse som gjorts på sju olika programmeringsspråk av L. Prechelt **[8]**.

2.2.1 C++

C++ är ett programmeringsspråk ämnat för allmänt ändamål **[9]** och är ett kompileringsspråk **[8]**. Detta betyder att program skrivna i C++ behöver kompileras [10]till ett format som en dator kan förstå innan koden kan exekveras, maskinen behöver även rätt arkitektur för att programmet ska fungera **[10]**. C++ saknar dock vissa inbyggda verktyg som finns till exempelvis Python. C++ har exempelvis inget inbyggt bibliotekshanteringsverktyg. Utan bibliotekshanteringsverktyg måste bibliotek laddas ner, installeras on länkas manuellt, detta kan förbruka tid som skulle kunna användas mer produktivt. Anledningen för att C++ valts som ett av språken är för att som Prechelt visar **[8]** så har C++ potentialen att vara väldigt tidseffektivt vilket är önskvärt när stora mängder data ska behandlas. Den nya koden som skrivs i detta projekt kommer huvudsakligen vara i C++ med standarden C++ 17.

Vcpkg

Det kan vara krångligt att installera och koppla bibliotek till en IDE, det måste vara rätt version, rätt operativsystem och det måste även länkas internt. Eftersom C++ inte har ett inbyggt verktyg för bibliotekshantering så används verktyget *Vcpkg* för att undvika dessa problem. Vcpkg är ett bibliotekshanteringsverktyg för C och C++ som är kompatibelt med Windows, MacOS och Linux. Vcpkg är utvecklat av Microsoft och de lägger till bibliotek till verktyget när dessa är tillgängliga. De installerade biblioteken blir automatiskt kopplade till Visual Studio och endast en inkludering i utvecklingsmiljön är nödvändig.**[11]**

Libxl

LibXl **[12]** är ett bibliotek för hantering av .xls- och .xlsx-filer. I detta projekt används testversionen av LibXl för att läsa in mätdata som finns tillgänglig från de tidigare projekten. Det är inte meningen att dessa filtyper ska användas för inläsning i praktiken utan används för att testa inmatning av regulatordata till databasen. Testversionen av LibXl kan endast läsa 300 celler och kan inte läsa den första raden i en fil **[13]**. LibXl-3.9.4.3 av är versionen som använts under detta projekt.

Libpqxx

Libpqxx är det officiella biblioteket för hantering av PostgreSQL-databaser i C++. Biblioteket använder en BSD-licens vilket gör det okej att ladda ner, inkludera i en produkt samt sälja den. Biblioteket har de funktioner som krävs för att kommunicera med en PostgreSQL databas. Version 7.3.1 används i detta projekt.**[14]**

2.2.2 Matlab

Matlab är en produkt utvecklad av MathWorks som kombinerar en skrivbordsmiljö för analys och design av processer med ett programmeringsspråk med fokus på simulering, grafer, matriser och matematiska funktioner. Matlab kommer med en redigerare där kod kan skrivas och uppdateras i realtid som sedan kan kombineras med annan kod.**[15]**

För att använda Matlab krävs det en licens. Vid skrivandet av denna rapport kostar en enskild licens av standardversionen av Matlab 8600 kronor för en årlig licens som måste förnyas när den går ut. Alternativt finns det även en evig licens som kostar 21500 kronor, dock så tillkommer det en årlig kostnad för Matlabs underhållstjänst för programvaran.**[16]**

Det finns flera tillägg för att utöka Matlabs grundfunktionalitet som är skapade av MathWorks eller av Matlabs användare. Dessa tillägg går att ladda ner och installera via Matlab-applikationen. För mer information hur detta går till se **[17]**.

2.2.3 Python

Python är ett interpreterat programmeringsspråk ämnat för allmänt ändamål **[18].** Det som menas med att Python är interpreterat är att istället för att koden översätts till maskinkod innan exekvering som i ett kompileringsspråk så sker översättningen under

tiden som programmet exekveras, vilket betyder att koden går att exekvera på alla maskiner med interpretatorn installerad **[10]**. När Python installeras på en maskin medföljer även verktyget pip som används för att installera och hantera tillägg till Python som kallas *paket*. I detta projekt används Python version 3.8.0.

Python har valts som ett av programmeringsspråken för detta projekt för att med rätt paket, exempelvis NumPy **[19]**, går det att efterlikna den funktionalitet som finns i Ernfjäll och Larssons program.

Pandas

För att utföra tester och för att hantera data i Python har biblioteket pandas använts på grund av dess snabba och effektiva dataobjekt som kan ändras flexibelt. Pandas håller också en hög prestanda när olika dataset ska kombineras. Pandas har också mycket av sin kritiska kod skriven i Cython eller C för att göra biblioteket snabbare.**[20]**

Psycopg2

Psycopg2 är ett paket till Python för att hantera PostgreSQL-databaser **[21]**, **[22]**. Psycopg2s kommunikation med databasen är trådsäker och är designad för att användas av multitrådade program **[21]**, **[22]**. Psycopg2 används i detta projekt för att populera databaser inför tester samt för att jämföra inläsningen av data i olika programmeringsspråk. Versionen av psycopg2 som används i detta projekt är version 2.8.6.

2.2.4 Visual Studio

Visual Studio 2019 Community är en IDE skapad av Microsoft. Visual Studio är en IDE som är passande för programmering i bland annat C++. Visual Studio har tillägg för bland annat databashantering, mobil utveckling och spelutveckling. Visual Studio används i detta projekt för skrivandet av program i C++. För mer information om Visual Studio se **[23]**.

2.2.5 Visual Studio Code

Visual Studio Code är textredigerare skapad av Microsoft. Det finns tillägg till Visual Studio Code för att efterlikna funktioner av en IDE, såsom kompilering, avlusning, testning och exekvering. I detta projekt har Visual Studio Code använts för att skriva kod i Python. För mer information om Visual Studio Code se **[24]**.

2.2.6 FreeOpcUa

För kommunikation via OPC används biblioteket FreeOpcUa för C++. FreeOpcUa är ett open source projekt för implementation av standarden OPC-UA och finns tillgängligt för C++ och Python **[25]**. FreeOpcUa finns tillgängligt för nedladdning på GitHub här **[26]**.

2.2.8 PostgreSQL

PostgreSQL är ett verktyg för att hantera relationsbaserade databaser via SQL. I detta projekt används PostgreSQL för att spara information som kommer in via OPC-klienten. PostgreSQL kommer med ett CLI-program för att övervaka databaser och det finns även ett GUI-program som heter pgAdmin med samma funktionalitet.

PostgreSQL har inbyggd funktionalitet för något som kallas *"triggers"* som är ett kommando att databasen automatiskt ska utföra en specifik funktion när en viss typ av operation utförs. En *"trigger"* kan kopplas till en *"view",* i princip en namngiven databasförfrågan. Detta för att utföra en viss operation som inte annars skulle gå att utföra före eller efter en förfrågan, exempelvis att lägga till eller ta bort information som inte ingår i angiven databasförfrågan. För mer information se **[27]** för *"triggers"* och **[28]** för PostgreSQl.

pgAdmin

PgAdmin är postgres GUI-program för att diagnosera, kontrollera och utveckla databaser. I pgAdmin kan man i realtid se användningen av olika databaser och servrar representerat grafiskt. Det går även att se aktuella kopplingar till databasen i verktyget vilket kan hjälpa med diagnosering av programkod. PgAdmin är mycket hjälpsamt om användningen av PostgreSQLs CLI-verktyg inte föredras, eftersom det är enkelt att sätta upp relationerna i en databas med hjälp av pgAdmins "*Query Tool"*. För mer information om pgAdmin besök **[29]**.

Structured Query Language

"Structured Query Language", förkortat SQL, är ett programmeringsspråk som skapades under 1970-talet av IBM **[3]**. SQL används inom kommunikation med relationsbaserade databaser för att bland hämta, lägga till, uppdatera och radera data och tabeller i en databasens. För mer information om hur SQL fungerar med PostgreSQL se dokumentationen här **[30]**.

2.3 Hårdvara

Under projektet används två persondatorer för skrivande och test av programkod. Nedanför finns information om datorernas processorer, systemminne och lagring. Datorernas specifikationer är:

Dator-1

- **Processor**: Intel i5-9600K 3.70 GHz, sex kärnor, sex logiska processorer
- **Systemminne**: 16 GB, DDR4, 2666 MHz
- **Lagring**: NVMe PCIe m.2 SSD, 1 TB, 3480 MB/s läsning, 3000 MB/s skrivning

Dator-2

- **Processor**: Intel i5-4670K 3.40GHz, fyra kärnor, fyra logiska processorer
- **Systemminne**: 16 GB, DDR3, 2400 Mhz
- **Lagring**: V-NAND Sata SSD, 500 GB, 550 MB/s läsning, 520 MB/s skrivning

3 Metod och Genomförande

Projektets genomförande har delats upp i olika delar som beskrivs nedan. Själva metoden för utförandet av projektet kan beskrivas som en variant av testdriven utveckling där ett mål har funnits och olika tester har utförts för att komma fram till det resultat som var bäst löste problemet. Själva samarbetet har skett på distans med hjälp av Zoom- och Discord-möten. Koden har funnits upplagd på Github för versionshantering. Delarna för genomförandet är:

- Del 3.1 handlar om grundläggande val om språk.
- Del 3.2 handlar om databasen och dess konstruktion.
- Del 3.3 handlar om implementation av OPC-servern och OPC-klienten.
- Del 3.4 handlar om sortering och analys av data.

3.1 Grundläggande om Programmeringsspråk

C++ har använts i detta projekt primärt för utveckling av servern och klienten som skulle kommunicera via OPC. Det har även använts för att undersöka möjligheten av att vidareutveckla analysprogrammet i C++.

Python har använts i projektet för att möjliggöra testandet av databaserna på ett enkelt sätt. Det har även använts så att resultat från Python kunde jämföras med Matlab för att undersöka möjligheten om Python är kapabelt att ersätta Matlab för vidareutveckling av programmet.

PostgresSQL har använts för att sätta upp databaserna och dess API har använts för att hämta och skicka information till databasen.

3.1.1 Bibliotek till C++

Kriterierna för att välja biblioteken var att de skulle lösa ett specifikt problem, exempelvis uppkoppling till databas, och att de skulle ha tillräcklig dokumentation för att kunna använda bibliotekets komponenter utan att behöva undersöka källkoden. Efter ett bibliotek hade hittats för en viss funktionalitet, testades det för att se om biblioteket faktiskt kunde uppfylla den uppgiften som det valdes för. För att länka biblioteket till Visual Studio användes Vcpkg eller manuell länkning.

Manuell Länkning

För att kunna använda vissa bibliotek i C++ behöver bibliotekets källkod hämtas och kompileras för den maskin som ska använda det, och sedan länkas till den IDE som ska användas. Denna manuella kompilering och länkning har gjorts i Visual Studio då det användes för C++ i projektet. I detta projekt har manuell länkning endast använts för biblioteket LibXl.

3.1.2 Bibliotek till Python

Bibliotekshanteringen i Python sker med hjälp av pip som hämtar och installerar det specifika bibliotek som anges. Det är därför mycket simpelt att hantera bibliotek i Python och om något inte skulle fungera är det möjligt att avinstallera bilbliotek med pip eller ändra version av biblioteken.

3.1.3 Val av Integrerad Utvecklingsmiljö

Visual Studio 2019 Community valdes som utvecklingsmiljö för C++ på grund av att Visual Studio har integrerat stöd för projekt skrivna i C++. Det var även enklare att importera och bygga projekt i Visual Studio än i alternativet som övervägdes, Visual Studio Code.

Visual Studio Code valdes för att skriva kod i Python då den inbyggda funktionaliteten kombinerat med tillägget för Python täckte de behov som fanns för projektet. Tillägget för Python inkluderar användbara funktioner såsom felsökning, kodnavigering, kodformatering, och refaktorering.

3.2 Databas

För att frångå att använda lösa filer för att lagra data inför analys implementerades en databas via PostgreSQL. Detta avsnitt beskriver processen för hur denna databas har konstruerats. Detta inkluderar sådana saker som vilka val som gjorts för vilka tabeller och relationer databasen skulle innehålla samt hur databasen ska reagera när ny information läggs till. Några viktiga upptäckter som var relaterade till databasen tas också upp i avsnittet.

3.2.1 Design och Konstruktion

Innan databasen skapades så konstruerades ett ER-schema över dess tabeller och relationer utifrån den data som fanns tillgänglig från det tidigare examensarbetet av Ernfjäll och Larsson. Den information som fanns tillgänglig bestod utav grundläggande information om regulatorerna och även information hämtad från regulatorer i drift. Informationen från regulatorer i drift var: ärvärde, börvärde, styrsignal och vilket läge regulatorn var i. Utifrån detta valdes att alla mätvärden för en regulator i en viss tidpunkt skulle lagras tillsammans och att informationen om regulatorerna skulle lagras tillsammans i samma tabell.

Utöver den data som skulle lagras övervägdes även framtida användning av databasen under designfasen. När databasen är i drift så kan det vara användbart att ha ett arkiv av tidigare mätvärden för att analysera trender över en längre period för att hitta problem som uppstår regelbundet. Av denna anledning valdes det att mätdata från regulatorerna skulle lagras i två tabeller, en som hade alla mätvärden och en som var ämnad att bara ha värden inför nästa analys.

När dessa val gjorts och ER-schemat var färdigställt så togs beslut om och hur *”triggers”* skulle användas när information infördes i databasen. För grundläggande

information om regulatorerna så togs beslutet att uppdatera den nuvarande regulatorn vid ett försök att lägga till en ny regulator med samma namn. Då felvärde för regulatorerna inte fanns i den data från tidigare arbeten så skrevs det en *”trigger”* för att räkna ut detta från ärvärdet och börvärdet när ett regulatortillstånd läggs till samt spara detta tillsammans med övrig mätdata.

När designen var klar skrevs koden för tabeller och *”views”* i en .sql-fil och koden för *”triggers”* i en annan. Dessa importerades till databasen via pgAdmins *”Query Tool”*. En grafisk representation av många databas komponenter kan ses i bilagor 6-9 och koden för databasens tabeller och funktioner kan ses i bilagor 10-11.

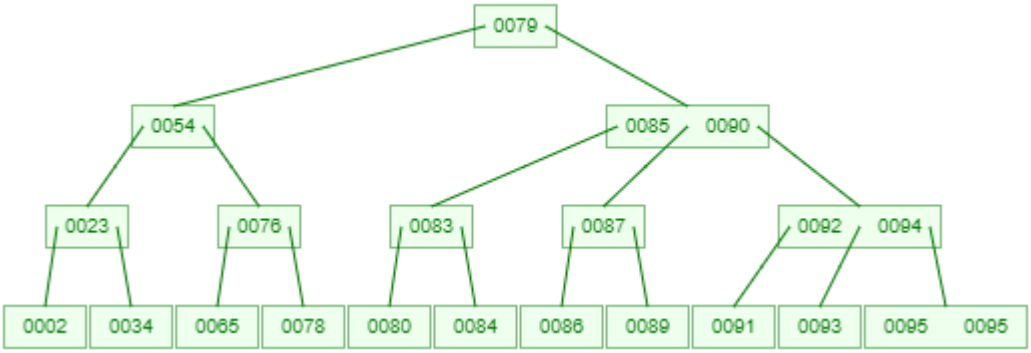
3.2.2 Upptäckter under Utvecklingen - B-tree

Ett problem upptäcktes vid ett test utav en databas att inläsningen från tabellen med tillstånd tog betydligt mer tid än förväntat när antalet regulatorer ökade. För att motverka detta infördes ett index på en kolumn för att optimera sökning där denna kolumn är inblandad. Att skapa/ lägga till ett index till en kolumn är inte gratis prestandamässigt eftersom kolumnen som indexet ligger på blir kategoriserat och läggs in i ett träd. Kostnaden beror på när indexet läggs till och hur mycket data som hanteras, detta har uppmärksammats när index har tagits bort och lagts till på redan skapade databaser med många tillstånd och regulatorer.

Efter en undersökning om vilka sorters index som kunde införas till databasen stod det mellan B-tree eller ett hash-index. Det visade sig att hash-indexet inte presterade bättre än B-tree men tog längre tid att införa till databasen, därför valdes B-tree att användas i projektet.

B-tree-indexet fungerar på alla jämförelser med kolumnen där “=”, ”<” eller “>” används medan hash-indexet bara fungerar på operatorn ”=”. Om ett index läggs till efter att värden har lagts in i databasen måste alla värden gås igenom och sorteras enligt indexets struktur som visas i Figur 3.1. Om indexet istället läggs till innan värdena så sorteras värdena när det läggs in i databasen och detta tar en marginell extra tid.

Figur 3.1 visar strukturen av ett B-tree och det huvudsakliga indexet som används i databasen. Det fungerar genom att medianvärdet alltid försöker ligga i roten och trädet flyttar runt värden när nya värden kommer in. Roten kan ändras med tiden och inga extremt skeva träd kan uppstå.



Figur 3.1: *Struktur av ett B-tree.* ***[31]****,* ***[32]***

3.3 Server och Klient

Detta avsnitt går igenom utvecklingsprocessen för servern och klienten. Båda av dessa var designade att kommunicera via en koppling med standarden OPC-UA. Detta avsnitt kommer också gå igenom några av de svårigheter som uppstod under denna process och hur de löstes.

3.3.1 Prototyp av Databaskoppling

För att hämta data från en server med regulatordata och sedan införa detta i en databas så skapades ett program i C++ med två delar: en del som läste data och en del som matade in detta i en databas. Den första versionen av detta program som skapades hade ingen förmåga att kontakta en server utan hämtade regulatordata direkt via LibXl från en .xlsx-fil med data från de tidigare examensarbeten. Efter att denna data hämtats så lades den sedan till i databasen med ett kort tidsintervall. LibXl har inte använts till något annat utöver denna första version. För att koppla programmet i C++ till PostgreSQL-databasen så användes biblioteket libpqxx

Något som märktes när detta program sedan testades var att det uppstod tomma rader i databasen med jämna mellanrum. Detta var på grund utav de begränsningar som testversionen av LibXl har när det gäller antalet celler som kan läsas in. Detta löstes genom att dela upp data till flera .csv-filer istället och sedan skriva om sättet som C++ programmet hämtar denna data genom inbyggd funktionalitet i C++. Regulatordatan delades upp till sex stycken filer där fem var ett blad var av .xlsx-filen och en sista som bestod av alla regulatornamn som sedan användes för att hålla reda på för vilken regulator som ett hämtat tillstånd tillhörde.

Efter att detta testats för att se att inmatningen till databaser sker som det ska så delades detta första program upp i två delar som kommunicer via OPC-UA: en klient och en server.

3.3.3 Klient

Klienten som utvecklades utifrån delen från den tidiga prototypen som matade in data i databasen. Den existerande delen modifierades minimalt och majoriteten av denna del av arbetet var att skapa komponenten som hanterar kommunikationen med servern.

Ett problem som uppstod under denna del var att versionen av biblioteket FreeOpcUa för C++ har väldigt bristande dokumentation, vilket ledde till att klienten behövde utvecklas utifrån dokumentation för versionen av biblioteket för Python samt exempelkod skriven i C++ som fanns tillgänglig på GitHub **[26]**. Klienten designades för att kunna kontakta en databas och en server som ligger på andra maskiner.

3.3.4 Server

I början av projektet var det planerat att en OPC-server skulle skapas med hjälp av Chalmers resurser. Denna server skulle simulera en verklig industrimiljö med PID-regulatorer. Chalmers server kunde dock inte skapas i tid till arbetets slut, därför skapades en enkel serverprototyp istället som kan skicka data till en klient. Servern utvecklades vidare från den delen av prototypprogrammet som hanterade inläsning från .csv-filer.

Likt delen som hanterar databaskoppling så modifierades delen som sköter inläsning från .csv-filer minimalt. Återigen så var denna del inriktad på kommunikation via OPC och återigen så uppstod det problem på grund av den bristande dokumentationen av FreeOpcUa. Detta löstes på samma sätt som för klienten, vilket var att servern utvecklades från exempelkod i C++ och dokumentation för Python-versionen av FreeOpcUa. Eftersom servern var designad för att endast vara en prototyp så hårdkodades den med fast adress och portnummer.

3.4 Inläsning och Analys

Detta avsnitt går igenom processen bakom arbetet som gjorts för hämtning, sortering och analys av regulatordata i Matlab. Det går även igenom processen för att göra om programmet skrivet i Matlab till en installerbar applikation. Några bakomliggande detaljer angående hämtning av data från databasen gås även igenom.

3.4.1 Hämtning av Data

Matlab har tillgång till många olika funktioner för att hämta data från en databas. I detta projekt har funktioner designade för relationsbaserade databaser använts för att hämta data i ett format som är kompatibelt med analyskoden från Larsson och Enfjäll.

Matlabfunktionen *fetch* hämtar data från den anslutna databasen enligt given databasförfrågan. *Fetch* kan använda flera olika sorters anslutningsmetoder och den returnerar en tabell från databasen som innehåller de rader och kolumner som matchar de parametrar som finns i given databasförfrågan. *Fetch* är PostgreSQL-

baserad och kanske inte fungerar för andra databaser. *Fetch* var det snabbaste alternativet att använda när data skulle hämtas i detta projekt.

En annan funktion som testades var *read. Read*-funktionen läser inte data direkt från en databas utan läser data från en *databaseDatastore*. En *databaseDatastore* är ett objekt som innehåller resultatet av en databasförfrågan. En *databaseDatastore* använder samma sorts koppling som funktionen *fetch.* I jämförelse med *fetch* är kombinationen av *read* och *databaseDatastore* långsammare och kräver även fler steg för att utföra samma uppgift.

Andra funktioner som testades var *readall*, *select* och *sql* men det visade sig att det var antingen mer komplicerat att skapa en anslutning till databasen för dessa typer av anrop eller så var de långsammare för användningsområdet.

Fetch istället för Select

Select funktionen användes för att hämta data från databasen tidigt i projektet. Funktionen testades eftersom den används i exempel på Matlabs hemsida för att hämta data från en databas. *Select* ersattes sedan av *fetch* på grund av prestandaskillnader. *Select* använder också ett mer begränsat sätt att ansluta sig till databasen vilket kräver att speciella drivrutiner behövde laddas ner och länkas med en direkt väg till programfilen.

Senare byttes det till att använda *PostgreSQL Native Interface* **[33]** och *fetch* i Matlab istället och då behövdes endast databasens namn, användarnamn, lösenord och adress för att skapa en koppling till databasen. Denna information var hårdkodad under projektet men ändrades senare så att parametrarna kan ändras för att skapa en koppling till olika databaser vid start av programmet.

Formatering av Data i Matlab

För att kunna behålla så mycket funktionalitet som möjligt från de gamla examensarbetets kodbas har den data som hämtas ställts upp på samma sätt som det gjordes i de gamla projekten. Programmet modifierades så att data hämtas från en databas istället för en Excel-fil. Den inlästa datan sorteras på samma sätt och lagras i en *struct* som sedan sparas i en .mat-fil, vilket är en filtyp ämnad för Matlab, för att sedan användas i analysdelen av programmet.

Inläsning av Irreguljär Datamängd från Databas

Under skapandet av tester prövades det vad som skulle hända om regulatorerna hade olika många tillstånd att hämta från databasen, vilket skulle kunna hända om en regulator samplar data snabbare eller långsammare än andra.
Det visade sig att koden inte kunde hantera detta då tabeller användes för att spara den data som hämtas från databasen. Tabeller i Matlab tillåter inte ihopsättning av tabeller med olika storlekar. Detta kunde lösas genom att representera den sparade data som en array istället för en tabell. Detta array-objekt sparas sedan i en *struct* som senare används för att sortera den data som skall skickas till analysen, det hade

fungerat lika väl att spara arrayen i en annan array eller en vektor men *structs* valdes då de växer dynamiskt i Matlab på ett smidigt sätt.

Test av Parallell Inläsning från Databas

Ett av de sista sakerna som testades med inläsning från en databas i Matlab var att använda Matlabs parallella for-loop, *parfor*, för att hämta data från databasen. Detta testades då det märktes i pgAdmin att det var ett par sekunders mellanrum mellan hämtningar av data. Under vanlig drift märktes det även att antalet transaktioner per sekund var ungefär 20, för att se grafer för transaktioner per sekund se bilagor 3 och 4.

3.4.2 Ändringar i Analyskoden

Huvuddelen av det arbete som skett med analyskoden har varit fokuserat på delen som sköter inläsning av data, men det var några ändringar som utfördes.

I den gamla koden upptäcktes det att felvärdet som gavs till regulatorer av typen *LC* på grund av att den data som användes för att räkna ut felvärdet togs från fel regulatortyp, detta ändrades så att data hämtades från *LC*-listan.

Den ursprungliga koden sparade inte svartlistan av vilka regulatorer som var felaktiga utan skrev bara ut detta i terminalen. Detta ändrades så att svartlistan istället sparas i en .csv-fil.

I ett test av programmet märktes det att skapandet av svartlistan hade en tendens att krascha när genererade testdata användes. Detta visade sig var för att koden försökte använda ett fält i en *struct* som inte alltid fanns när den försökte hitta en regulators position i svartlistan. Den första lösningen på detta problem var att kontrollera om fältet fanns innan det skulle användas, men detta ledde till att exekveringstiden ökade. Den slutgiltiga lösningen var att regulatorernas positioner i svartlistan sparades i en *map* där regulatorns namn var kopplat till ett index.

3.4.3 Application Compiler

Application compiler är ett tillägg för Matlab som för att generera en installerbar applikation utav program skrivna i Matlab. *Matlab Runtime* som behövs för att köra Matlabapplikationer medföljer i installationsfilen. *Application compiler* har testats för att se hur en applikation skriven i Matlab kan användas utan att användaren behöver en licens. När det prövades första gången visade det sig att versionen av kompilatorn som gick att ladda ner inte var kompatibel med versionen av Matlab som användes i arbetet, vilket var version 2019b. Detta ledde till att nerladdningen kraschade konstant. Efter felsökningen så löstes problemet genom att ladda ner Matlab version 2021a och att installera om alla tillägg som användes i programmet.

För att kunna använda *Application compiler* krävdes det att lägga till en *main*-funktion där programmet startar. I detta arbete gjordes detta i filen som hanterade anslutningen

till databasen. Alla funktioner som sedan används av *main*-funktionen lägger Matlab till automatiskt.

För att tillåta applikationen att vara mer flexibel gällande vilken databas som ska användas lades det till inparametrar till *main*-funktionen. Dessa inparametrar var databasnamn, användarnamn och lösenord till databasen.

3.5 Konvertering av data

I projektet försöktes konvertering av datatyper att undvikas då det är ett extra steg i processen som tar mer tid. De konverteringar som inte kunde undvikas sker när klienten hämtar data från servern då vissa strängar måste konverteras till nummer innan de kan läggas till i databasen. Denna konvertering var nödvändig då strukturer inte kunde skickas med hjälp av det valda OPC-biblioteket utan vektorer av strängar behövde användas för att skicka information. I Matlab-koden har konvertering också försökt att minimeras. De enda konverteringar som inte kunde undvikas i Matlab är när olika datastrukturer inte var kompatibla med vissa funktioner.

3.6 Testning

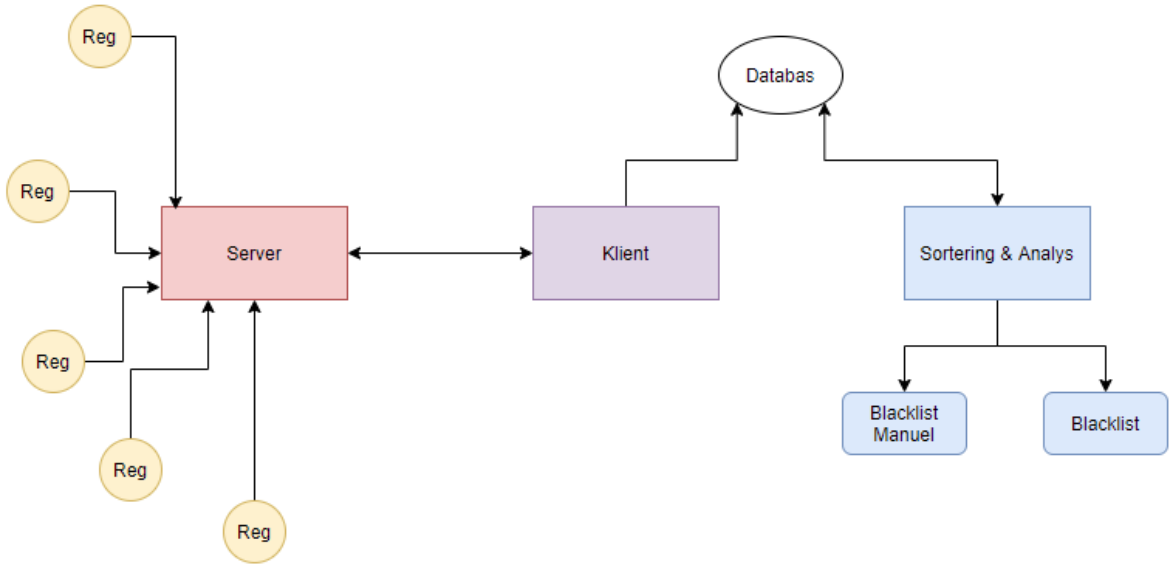
Tester på OPC-kopplingen har utförts genom att servern till början hade tillgång till data för 100, 1000 och 10000 regulatorer med 3600 tillstånd per regulator. Därefter kördes servern och klienten för att testa om databasen kunde matas med uppdateringar för alla regulatorerna varje sekund. Efter att det blev uppenbart att 10000 regulatorer inte klarade detta krav gjordes fler tester med färre regulatorer för att hitta gränsen för att kopplingens skulle klara av alla regulatorer under en sekund på Dator-1.

Inläsningen från databasen har testats i Matlab, C++ och Python. Alla tester har utförts tre gånger var för 100, 1000 och 10000 regulatorer med 3600 tillstånd per regulator. Dessa test utfördes för att testa hur lång tid det tar att hämta all information samt hur mycket arbetsminne som processen använde. Inläsningen skedde även i den gamla Matlab-koden med samma data för jämförelse.

Slutligen utfördes tester för hela inläsning- och analysprocessen i både den gamla och nya Matlab-koden med samma data som i testerna för enbart inläsning. Även i detta fall utfördes testerna för att utforska exekveringstid och minnesanvändning. Testerna utfördes tre gånger var för varje uppsättning regulatorer i båda Matlab-programmen.

4 Resultat

Projektet är byggt på en server som får in data från PID-regulatorer, denna data hämtas från .csv-filer. När servern får ett anrop från klienten hämtar den en uppsättning tillstånd och skickar datan till klienten via en OPC-kopplingen, denna data sparar sedan klienten i en relationsbaserad databas som analysprogrammet kan hämta information från. Efter att sorteringen och analysen har skett skapas två svartlistor, en för regulatorer i manuellt läge och en för regulatorer som något kan vara fel med. Figur 4.1 är en illustration av de olika delarna av projektet och hur de hänger ihop.



Figur 4.1: *De olika komponenterna av projektet.*

4.1 Databas

Databasen som har skapats i detta arbete består utav tre tabeller och två relationer, de tre tabellerna är: *Regulators*, *States* och *StatesArchive*. Utöver detta så innehåller även databasen två *”views”* och två ”*triggers*”. Databasen är utformad efter den data som finns tillgänglig från de tidigare examensarbeten. Koden som används för att att skapa tabeller, *“views”* och *“triggers”* har delats upp i två filer som kan användas i PostgreSQLs CLI-program eller i pgAdmin, en för tabeller och *“views”* och en för *“triggers”,* se bilagor 10 och 11 för denna kod.

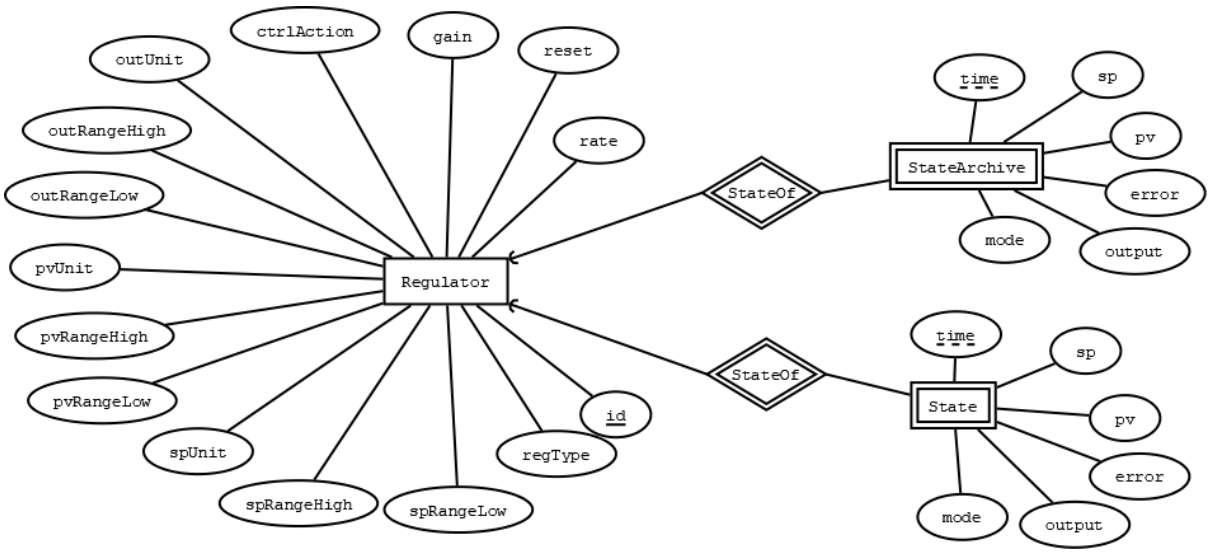
4.1.1 Tabeller och Relationer

Tabellen *Regulators* innehåller all information om regulatorernas egenskaper. Informationen som lagras i *Regulators* är: regulatorns namn (id) som än nyckeln för tabellen, regulatorns typ (regType), min- och maxvärden samt enheter för regulatorns olika signaler, *”controll action”* (ctrlAction) som anger hur utsignalen ska förändras gentemot ärvärdet, *”gain”* som används i den proportionella delen av PID, *”reset”* som

används i den integrerande delen och *”rate”* som används i den deriverande delen. Se den vänstra delen av Figur 4.2 för exakt uppbyggnad av tabellen *Regulators*.

De två andra tabellerna *States* och *StatesArchive* är identiska i uppbyggnad och båda har likadana relationer till tabellen *Regulators*, men de har olika användningssyften. *States* är ämnad för att endast innehålla mätvärden inför nästa analys och efter att analysen så ska de använda mätvärdena tas bort, kolumnen för regulatornamn har även ett *B-tree*-index. Att mätdata raderas har dock inte implementerats men det går att hämta data ur en tabell samtidigt som den raderas. *StatesArchive* är ämnad för att innehålla historiska mätvärden som ett arkiv, detta då det kan vara användbart för att undersöka trender i mätvärden.

Dessa tabeller innehåller regulatorernas tillstånd vid en viss tidpunkt och är så kallade *”weak entities”*. *States* och *StatesArchive* innehåller: regulatornamn och tidpunkt som en sammansatt nyckel, börvärde, ärvärde, felvärde, utsignal och regulatorns driftläge. Den relation som båda har med *Regulators* är att regulatorn som ett tillstånd är för måste finnas i *Regulators*. Se den högra delen av Figur 4.2 för exakt uppbyggnad av tabellerna *States* och *StatesArchive*.



Figur 4.2: *Uppbyggnaden av tabeller och relationer i databasen.*

4.1.2 Views och Triggers

Databasen har två *“triggers”* med en *“view”* var, en för regulatortabellen och en gemensam för tabellerna med tillstånd. Dessa *“views”* hämtar alla information som finns i *Regulators* eller *States* och används endast för att de två *“triggers”* ska kunna köras som *“INSTEAD OF”*.

Den första *“trigger”* används för inmatning av data i *Regulators*, det den gör är att om regulatorn redan finns så ska de värden som finns i tabellen för den uppdateras till de nya värdena och om den inte finns så ska värdena läggas till som vanligt.

Den andra *“trigger”* är för inmatning av regulatortillstånd. Denna *“trigger”* har tre uppgifter: att räkna ut felvärdet som differensen mellan börvärde och ärvärde, att lägga till en tidsstämpel och att lägga till tillståndet tillsammans med felvärde och tidsstämpel i *States* och *StatesArchive*.

4.2 Server och Klient

Detta avsnitt behandlar servern och klienten som är skrivna i C++. Båda använder biblioteket FreeOpcUa för att hantera kommunikation med varandra. Detta avsnitt täcker också testresultatet för hur många regulatorer som kopplingen kan hantera inom en sekund. Klient- och server kopplingen har endast testats när båda är på samma maskin.

4.2.1 Server

Servern är endast en prototyp som används för att testa att OPC-kopplingen fungerar och den kan inte kopplas till verkliga regulatorer för att hämta information i realtid utan sex .csv-filer med data används istället. Servern består utav två delar, en som läser in data från filerna och en som hanterar kommunikationen med klienten. Endast en koppling har testats åt gången.

När Servern startar så läser den in alla regulatornamn och sparar dem i en vektor. Efter detta så väntar servern på ett anrop från en klient. Servern har tre funktioner som klienten kan kalla på: *getRegulators*, *getStates* och *canFetch*.

När funktionen *getRegulator* kallas så öppnar och läser servern filen med alla regulatorinställningar och lägger in detta i en tvådimensionell vektor av strängar. Efter all data lästs så stängs filen och servern skickar sedan den tvådimensionella vektorn till klienten som kallade på funktionen.

Funktionen *getStates* läser in en rad var från filerna som innehåller börvärden, ärvärden, utsignaler och driftlägen, när detta görs flyttas även filpekaren framåt. Sedan så kopplas ett värde från varje rad ihop med det regulatornamnet som har samma position i vektorn som värdet har i raden. Dessa sätts ihop till en vektor av strängar som sedan läggs in i en annan vektor. Servern skickar sedan vidare den tvådimensionella vektorn till klienten.

Funktionen *canFetch* kollar om det går att hämta nya tillstånd för regulatorerna, om det går så returneras en boolean med värdet *sant* annars så returneras *falskt*. Detta värde skickas sedan tillbaka till klienten.

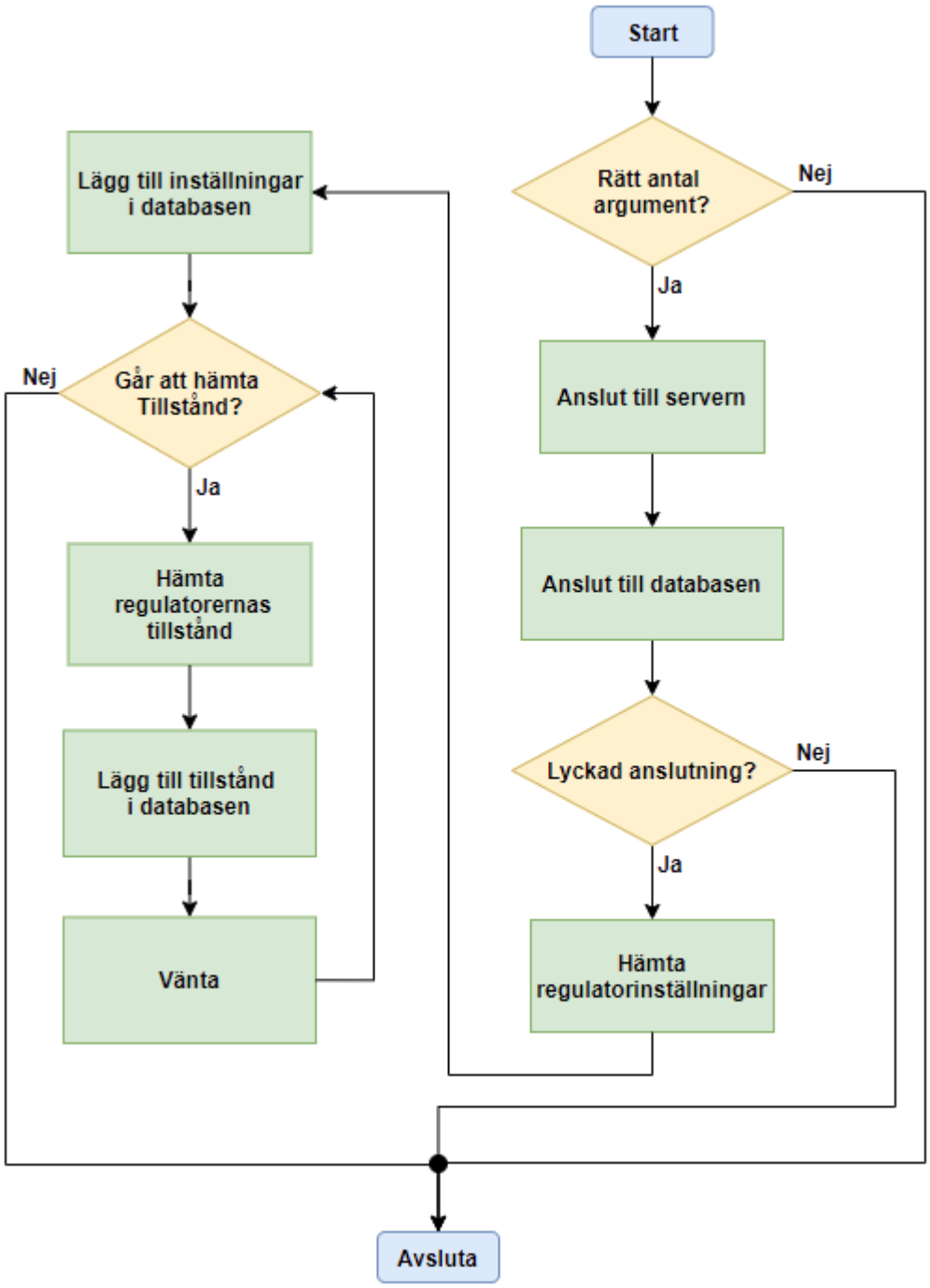
4.2.3 Klient

Klienten är ett program som består av två delar, en del hanterar kommunikationen med servern och den andra hanterar kommunikationen med databasen. Klienten använder biblioteket FreeOpcUa för kommunikation med servern och libpqxx för kommunikation med databasen. Klienten behöver sex argument vid uppstart för att kunna koppla sig till databasen och servern. Dessa argument är: databasens namn, användarnamn, lösenord, databasens adress, databasens port och adressen för OPC-servern.

Figur 4.3 på nästa sida visar programflödet för klienten. När klienten startar så kontrollerar den att rätt antal argument har angetts, om inte så skrivs det ut vilka argument som krävs och i vilken ordning, sedan avslutas programmet.

Om rätt antal argument finns så försöker klienten koppla sig till servern och databasen, om detta misslyckas så skrivs det ut ett felmeddelande och programmet avslutas. Om koppling till server och databas lyckats så hämtas alla regulatorinställningar från servern via server-funktionen *getRegulators* och dessa läggs sedan till i databasen.

Efter att inställningarna har skickats till databasen så går klienten in i en loop som ska exekveras en går i sekunden. Först sparas den nuvarande tiden i en variabel. Sedan kontrollerar klienten om det går att hämta tillstånd genom att kalla funktionen *canFetch* hos servern, om värdet är falskt så avslutas loopen och programmet. Om det går att hämta regulatortillstånd så görs detta med server-funktionen *getStates.* Tillstånden läggs sedan till i databasen och tiden när detta var klart sparas. Om differensen mellan starttiden och sluttiden är större än en sekund så börjar nästa iteration av loopen direkt annars så väntar den tills en sekund efter starttiden för att börja nästa iteration.



Figur 4.3: *Programflöde för klienten.*

Maxantalet av regulatorer som kan skicka data samtidigt utan att överskrida ett ensekundsintervall är mellan 2000 och 2500 regulatorer på Dator-1, vid 2500 börjar tidsintervallet att drifta märkbart (se Tabell 4.1). Detta är dock inget problem för det är möjligt att sätta upp flera servrar och klienter som pratar med samma eller olika databaser.

Tabell 4.1: *Tidsintervall i sekuner (s) mellan inmatning av status för regulatorer i databasen. Test utfört på Dator-1.*

Antal regulatorer	Tid (s)
100	1
1000	1
2000	1
2500	1.05
5000	2
10000	4

4.3 Inläsning och Analys

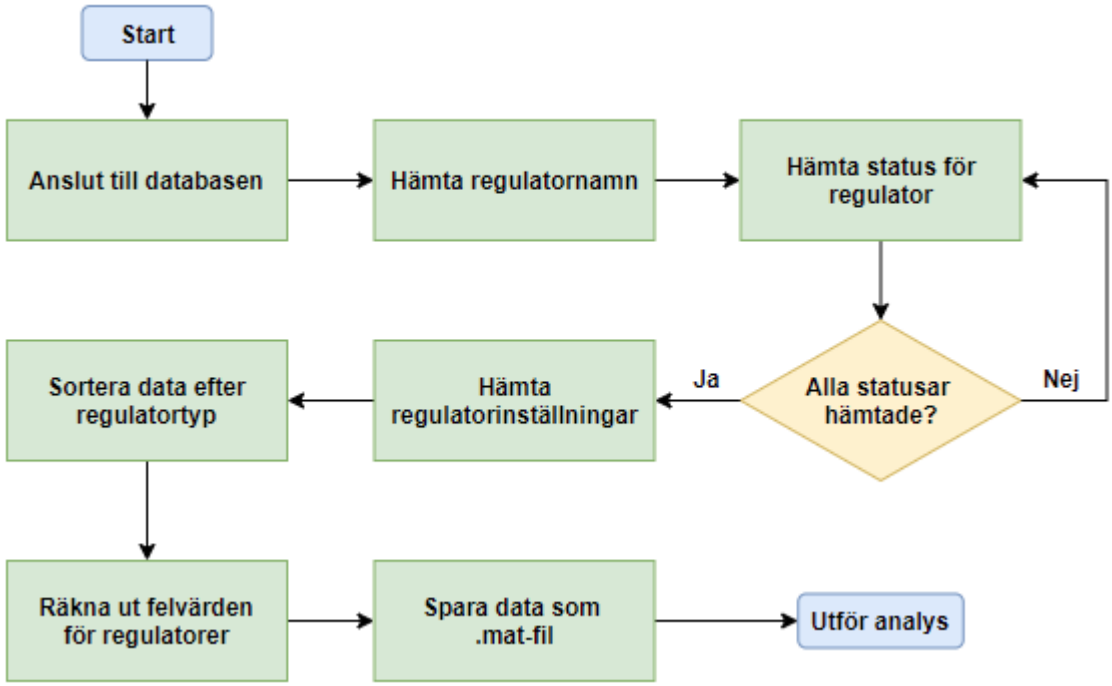
I stället för att skapa ett nytt program för analys och sortering så har tiden lagts på att förbättra det gamla Matlab-programmet och utöka dess funktionalitet. Många av ändringarna som gjordes har beskrivits i metoden men resultat var att sorteringen och hämtningen blev ungefär 98 procent snabbare för 100 regulatorer och ännu bättre när fler regulatorer används, data hämtas och analyserar nu också automatiskt en gång per timme med hjälp av en PostgreSQL-databas. En installationsfil har skapats för analysprogrammet som gör det möjligt att köra programmet utan en Matlab-licens, de drivrutiner som krävs för att köra en Matlab-applikation är inkluderad i installationsfilen.

4.3.1 Ändringar av Matlabkod

Den största ändringen är hur data läses in av analysprogrammet, i delen som hanterar inläsning av data använder nu en relationsbaserad databas. Programmet går nu även på ett schema där inläsning och analys utförs med ett timmes intervall. Programmet kräver nu tre argument för att koppla till databasen: databasnamn, användarnamn och lösenord. Figur 4.4 på nästa sida visar hur den nya inläsningsprocessen som används i Matlab fungerar. För jämförelse av Matlab, C++ och Python så används bara de stegen innan hämtning av regulatorinställningar.

Det första som görs i den nya processen är att ansluta till databasen och hämta namnen på alla regulatorer. Efter detta så loopar programmet igenom listan med namnen och hämtar alla tillstånd för regulatorerna som sedan lagras i en *struct*. När alla tillstånd har hämtats så hämtas även inställningarna för alla regulatorer. Efter att all regulatordata har hämtats så läggs dessa in i en *struct* sorterade efter regulatortyp. Nästa steg är att räkna ut felvärdet för varje regulator i varje tidpunkt. Detta finns I databasen som en del av regulatorernas tillstånd, men en kvarstående del i Matlabkoden från den gamla versionen är att felvärdet räknas ut i programmet då det

går snabbare än att hämta en extra kolumn från databasen. Detta finns kvar då det inte märktes någon skillnad i förbrukad tid gentemot att hämta detta från databasen. Det sista steget är att spara den *struct* med regulatorer som en .mat-fil.



Figur 4.4: *Programflöde för inläsningsprocessen.*

Analysdelen av programmet är nästan identiskt med den gamla versionen. Endast två modifieringar finns. Den första ändringen är att analysprogrammet nu sparar två svartlistor som .csv-filer, en för regulatorer i manuellt läge och en för regulatorer som flaggats som felaktiga.

Den andra ändringen är att programmet nu använder en *map* där en regulators namn är kopplat till ett index i svartlistan för felaktiga regulatorer. Den gamla versionen loopade igenom svartlistan och jämförde den positionen med regulatorn som var felaktig, vilket kraschade programmet när den genererade testdatan användes. Programmet kollar nu om regulatorns namn finns som nyckel, och om den gör det så hämtar den värdet som nyckeln är kopplat med och lägger till ett värde för regulatorn i svartlistan. Om nyckeln inte finns så läggs regulatorn till i slutet av svartlistan och regulatorn och indexet läggs till i *map*.

4.3.2 Testresultat

I Tabell 4.2 nedan finns resultatet för tider av inläsning med och utan index samt sortering av regulatorer i Matlab. Indexet är på regulatornamnen i tabellen *States* i databasen. I alla testfall hämtas endast 3600 regulatorer. I de två första testerna i Tabell 4.2 så fanns alla 55000 tillgängliga tillstånd per regulator i tabellen och i det fallen så hjälpte inte indexet förbättra inläsningstiden förutom för testet med 76 regulatorer på Dator-1. I alla andra tester så fanns endast 3600 tillstånd per regulator i tabellen och då gav användandet av index en stor förbättring av inläsningstiden för båda datorer. Jämförelsevis så är sorteringen ungefär dubbelt så snabb på Dator-1 som på Dator-2.

Tabell 4.2: *Jämförelse av inläsningstid i sekunder (s) från databas med index (MI) och utan index (UI) samt tid i sekunder (s) för sortering av data i Matlab. Kolumnen AT (Alla Tillstånd) betecknar om alla 55000 tillstånd för en regulator finns i databasen eller endast 3600. Utfört på båda maskiner.*

	Dator-2			Dator-1			
Antal regulatorer	**MI (s)**	**UI (s)**	**Sortering (s)**	**MI (s)**	**UI (s)**	**Sortering (s)**	**AT**
76	5.14	5.22	0.69	3.32	7.04	0.31	JA
152	25	26	1.09	26.58	27.16	0.68	JA
152	8.15	21.01	1.02	6.43	19.58	0.61	NEJ
228	12.49	29.29	1.62	9.67	32.6	0.83	NEJ
380	20	56	2.82	15.28	71.13	1.25	NEJ
760	39.60	168.51	5.51	32.94	150.17	2.6	NEJ
7600	375.98	>4612*	58.46	332.81	>7000*	28.66	NEJ

* Dessa tester avslutades då de överskred en timme som var den maximalt tillåtna tiden.

I Tabell 4.3 på nästa sida finns resultatet av jämförelse i Python för inläsning med och utan index på regulatornamnen i tabellen *States* i databasen. Som med Matlab så är inläsningen från databasen mycket snabbare när index används än om det inte gör det för alla tester där inte alla 55000 tillstånd används. När 55000 tillstånd används så är förbättringen marginell. Till skillnad från testet som utfördes i Matlab så inkluderades inte sortering i testet för Python.

Tabell 4.3: *Inläsningstider i sekunder (s) i Python med och utan index på regulatortillstånd. Kolumnen AT (Alla Tillstånd) betecknar om alla 55000 tillstånd för en regulator finns i databasen eller endast 3600. Utfört på Dator-2.*

Antal regulatorer	B-Tree (s)	Utan index (s)	AT
76	1.33	1.5079	JA
152	17.72	19.6385	JA
152	2.20	20.3730	NEJ
228	2.43	22.2579	NEJ
380	4.03	44.07802	NEJ
760	7.26	151.3350	NEJ
7600	78.54	>2000*	NEJ

* Testet avbröts efter denna tid.

Jämförelse av Inläsning i olika Programmerinsspråk

Den nya metoden för att läsa in regulatordata till analysprogrammet är att hämta data från en relationsdatabas istället för en xlsx-fil. Alla implementationer av den nya metoden i olika programmeringsspråk är snabbare än den gamla koden i de tester som utförts (se Tabell 4.4) och de lagrar i data internt med samma struktur för att vara så lika som möjligt. Inläsningstiderna för alla implementationer av den nya processen ökar linjärt (se Tabell 4.4).

Den gamla metoden tog 292.04 sekunder för 100 regulatorer och 1890.61 sekunder för 1000 regulatorer, testet för 10000 regulatorer avbröts dock i förtid då inläsningen tog över en timme, (se kolumn ‘Matlab - gammal’ i Tabell 4.4), och var därför utanför den tillåtna tidsgränsen då ett önskemål var att det skulle gå att utföra en analys varje timme.

Den långsammaste implementation av den nya metoden, C++, tog 5.88 sekunder för 100 regulatorer, 58.04 sekunder för 1000 och 572.85 sekunder för 10000, (se kolumn ‘C++’ i Tabell 4.4), vilket är inom det acceptabla tidsfönstret med god marginal för genomförandet av analys.

Implementationen som används i det nuvarande programmet är något snabbare än C++ med tiden 4.45 sekunder för 100 regulatorer, 39.79 sekunder för 1000 och 375.68 sekunder för 10000, (se kolumn ‘Matlab - ny’ i Tabell 4.4).

Den snabbaste implementation är i Python och är det överlägsna alternativet då den läser in 100 regulatorer på 0.88 sekunder, 1000 på 8.56 sekunder och 10000 på 80.4 sekunder (se kolumn ‘Python’ i Tabell 4.4).

Tabell 4.4: *Jämförelse av inläsningstid i sekunder (s) mellan olika programmeringsspråk. Utfört på Dator-1.*

Antal regulatorer	Matlab - gammal (s)	Matlab - ny (s)	C++ (s)	Python (s)
100	292.04	4.45	5.88	0.88
1000	1890.61	39.79	58.04	8.56
10000	> 3600*	375.68	572.85	80.4

* Testet avbröts då exekveringstiden överskred en timme som var det önskade tidsintervallet.

De två olika metoderna av inläsningen av regulatordata har stora skillnader i hur mycket minne som används av programmet, detta illustreras i Tabell 4.5. Det är viktigt att notera att den uppmätta minnesanvändningen i Matlab inte är exakt då Matlab saknar ett enkelt sätt att granska minnesanvändning, till skillnad från Python och Visual Studios avlusningsläge för C++. För Matlab är minnesanvändningen uträknad genom att använda Windows aktivitetshanterare för processer och sedan subtrahera använt minne före exekveringen från det maximala värdet under exekveringen.

Den gamla metoden att läsa data från en xlsx-fil allokerar ungefär 100 gånger mer minne än något av programmen som läser från en relationsdatabas. Detta är dock osäkert när det gäller inläsningen av 10000 regulatorer då det inte gick att allokera mer minne på systemet under testets utförande (se anteckningen i Tabell 4.5).

Det finns även skillnader i hur mycket minne som allokeras i de program som använder relationsdatabasen, (se kolumnerna Matlab - ny, C++ och Python i Tabell 4.5). Skillnaden mellan dessa är dock liten, men utav programmen så är inläsningen i C++ mest minneseffektiv.

Tabell 4.5: *Jämförelse av använt minne i megabyte (MB) under inläsning av regulatordata. Utfört på Dator-1.*

Antal regulatorer	Matlab - gammal (MB)	Matlab - ny (MB)	C++ (MB)	Python (MB)
100	1648	20	13	64.5
1000	11758	115	112	164
10000	13500*	1192	1100	1160

* Allt minne som var tillgängligt under testets gång.

Jämförelse av analys i Matlab

Undersökningen av analystiden mellan det gamla programmet samt det nya visar att de optimeringar som gjorts har förbättrat exekveringstiden märkbart. I testerna med 100 och 1000 regulatorer är den nya koden 15 respektive 16 procent snabbare, men förbättringen är tydligare vid 10000 regulatorer då den nya koden är 38 procent snabbare (se Tabell 4.6). Vid jämförelse av den nuvarande koden så växer analystiden linjärt i proportion till antalet regulatorer (se Tabell 4.6).

Tabell 4.6: *Jämförelse av analystid mellan den gamla och nya Matlab-koden. Utfört på Dator-1.*

Antal regulatorer	Matlab - gammal (s)	Matlab - ny (s)	Förbättring (%)
100	3.25	2.75	15
1000	25.87	21.68	16
10000	411.09	254.94	38

Analyskodens minnesanvändning är ungefär likvärdiga mellan den gamla och nya koden, (se Tabell 4.7). Som i fallet för minnesanvändningen under inläsning så är även värdena i Tabell 4.7 inte exakta av samma anledning, minnesanvändningen är även uppmätt med samma process. Minnesanvändningen för analysen kommer direkt efter att inhämtningen har slutförst och frigjort det minne som den processen använde.

Tabell 4.7: *Jämförelse av minnesanvändning under analys i den gamla och nya Matlab-koden. Utfört på Dator-1.*

Antal regulatorer	Matlab - gammal (MB)	Matlab - ny (MB)
100	56	50
1000	373	410
10000	3378	3570

Parallelprogrammering

Den nuvarande kodens sätt att hantera databaskopplingen är inte kompatibel med Matlabs parallellisering av loopar, koden behövde därför skrivas om inför testet så att varje iteration av loopen har sin egen koppling till databasen.

Resultatet av att försöka parallellisera inläsningen av data från databasen visar att på Dator-1 är prestandan av den nuvarande koden och en parallelliserad version ungefär likvärdiga i exekveringstid, (se kolumnerna ‘Matlab - ny’ och ‘Parallel - test’ i Tabell 4.8), med en genomsnittlig skillnad på 10.26 procent.

Jämförelsen mellan den parallella koden och samma kod exekverad sekventiellt (se kolumnerna ‘Sekventiell - test’ och ‘Parallel - test’ i Tabell 4.8), visar att anledningen till att den parallella koden inte är snabbare är de ändringar som behövdes för att tillåta att Matlabs parallella for-loop kan användas. Även med denna försämring i sekventiell exekvering så denna kod genomsnittligt 56.44 procent snabbare när den exekveras parallellt.

Tabell 4.8: *Resultat av parallellisering av inläsning från databasen. Kolumn "Parallel - test" har resultatet efter inläsning från databasen med flera parallella kopplingar. "Sekventiell - test" har samma kod för anslutningar som "Parallell - test" men utförs sekventiellt istället. Båda kolumnerna bör jämföras med "Matlab - ny" som utförs sekventiellt med endast en databaskoppling. Utfört på Dator-1.*

Antal regulatorer	Matlab - ny (s)	Parallel - test (s)	Sekventiell - test (s)
100	4.45	4.68	10.15
1000	39.79	42.96	103.61
10000	375.68	460.27	1067.35

5 Diskussion

Detta kapitel diskuterar programflödet och vad som skulle kunna förbättra det, hur programmet blev snabbare samt vilka för- och nackdelar som finns, vad som krävs för att programmet ska kunna köras på en dator och hur detta projekt skulle kunna leda till besparingar i stora och små företag samt sänka slöseri av resurser.

5.1 Programflödet

Projektet består av flera olika program som är sammankopplade antingen med anrop (server till klient med OPC) eller en databas (klient och analys). Server och klienten fungerar just nu bara på samma nätverk, om servern ska vara kontaktbar utifrån så måste den använda porten öppnas i nätverkets router. Om klienten ska kunna ansluta sig till en server som inte är på samma maskin måste rätt IP-adress samt portnummer anges i programmet, just nu används bara *localhost*.

Matlabprogrammet har ändrats så att den kan ansluta till vilken lokal databas som helst genom att ge argument när programmet startas. I detta läge görs detta i en kommandotolk vilket inte är särskilt användarvänligt. De svartlistorna som nu sparas som .csv-filer skulle kunna användas av något annat program för att grafiskt visa vilka regulatorer som har fel eller står i manuellt läge.

5.2 Prestandaförbättring

Eftersom ingen tidigare server eller klient fanns, kan ingen jämförelse med gammal information göras på denna punkt, men resultatet blev att 2000 regulatorer per server är fullt möjligt under ett ensekundsintervall, vilket skulle fungera i en mindre miljö. Detta är dock på en persondator och inte på en maskin som är ämnad för att hantera en industrimiljö. Då en sådan dator inte varit tillgänglig under arbetet så har det inte gått att testa serverns eller klientens prestanda på en sådan maskin.

Inläsningsprocessen för att hämta regulatordata från databasen har sett stora minskningar av minnesanvändning och tidsförbrukning. Hämtningshastigheten såg sitt första framsteg när en databas började användas i stället för excel-fil, eftersom vid inläsning av en excel-fil måste stora delar av filen sparas i minne innan någon inläsning till programmet kan börja. När en databas istället används hämtas alla tillgängliga data för en regulator åt gången från databasen.

Under projektet har det uppmärksammats att hämtning av data tar olika lång tid beroende på hur många regulatorer som ska hämtas samt hur mycket data som redan fanns i en tabell. Introduceringen av ett index på regulatornamnet i tabellen *States* gav inte nödvändigtvis någon prestandaförbättring när det fanns mer data i tabellen än som skulle hämtas. Därför bör data som hämtas från *States* tas bort när den inte längre är nödvändig för att inte försämra prestandan i längden

Parallellisering

Som vi har tagit upp i resultatet så har inläsningen testats att utföras parallellt istället för sekventiellt. I testet som utfördes blev resultatet inte bättre på Dator-1 med parallell hämtning, men detta var med kod som tog mer än dubbelt så lång tid när den exekverades sekventiellt än den nuvarande koden. Detta var dock på en processor med sex kärnor, vilket betyder att den parallella inhämtningen kan vara mer effektiv på en processor med fler än sex kärnor.

”Approaches to Speed up Data Processing in Relational Databases” av Y. Shichkina **[35]** går igenom ett antal olika sätt snabba upp databehandlingen i en relationsbaserad databas. Enligt **[35]** så är parallellisering ett lovande sätt att förbättra prestandan, men att en direkt applikation av parallella beräkningsmetoder på en databasförfrågan är ineffektiv och att det inte går att applicera på vissa typer av databasförfrågningar. Utifrån detta så är kanske parallellisering av inläsningen inte värt att utforska mer i framtiden.

Även om inläsningen inte skulle förbättras av parallellisering så kan analysen förmodligen förbättras av att parallelliseras. Analysen skulle kunna parallelliseras på flera olika sätt, två sätt diskuteras nedanför.

Det första är att dela upp analysen så att en tråd hanterar en regulatortyp, men detta har nackdelen att inte alla kärnor kanske används, till exempel med fyra typer och 16 kärnor. Det kan även finnas en stor variation av antalet regulatorer av varje typ, vilket kan leda till att ett par trådar blir klara mycket snabbare än andra.

Det andra alternativet är att dela upp den totala mängden data på de tillgängliga kärnorna så att varje tråd hanterar alla typer men, bara en del av alla regulatorer. Till exempel om det finns två typer av regulatorer med 200 respektive 100 regulatorer var och datorn har fyra kärnor så skulle det gå att dela upp data så att varje tråd hanterar 50 regulatorer av typ 1 och 25 regulatorer av typ 2. På detta sätt skulle alla trådar vara klara samtidigt och analysen skulle kunna bli upp till fyra gånger snabbare än om en tråd gjorde allt.

Hämta färre kolumner

Hämtning av data till Matlab är i nuläget det som tar mest tid i Matlabprogrammet och testet att förbättra detta med parallellisering misslyckades. Det märktes dock att när felvärdet inte togs med i hämtningen så blev inläsningen snabbare. Ett alternativ för att förbättra inläsningen skulle därför vara att kombinera värden i databasen till en kommaseparerad sträng eller ett *JSON*-objekt för att läsa in färre kolumner. *JSON* är är ett sätt att strukturera data inför datautbyte som inte är språkberoende **[34]**. Nackdelen med dessa metoder är att det blir mer arbete i programmet för att sedan separera värden från resultatet. Detta har inte testats på grund utav tidsbrist, men det kan vara värt att undersöka i fortsättningen.

5.3 C++

C++ var det språk som förväntades vara snabbast, men i de jämförelser som gjorts så var C++ det långsammaste alternativet. Det var även det alternativet som använde minst minne utav programmeringsspråken, men i alla fallen så var denna skillnad liten.

Det är osäkert varför C++ var så mycket långsammare än Matlab och Python då det inte fanns tid att undersöka detta. Anledningen till att C++ är långsammare kan vara att implementationen av inläsningen i C++ är väldigt ineffektiv, mer specifikt att kontakten med databasen skulle gjorts på ett annat sätt. Ett annat alternativ är att biblioteket libpqxx är långsamt på att hämta data på det uppdelade sättet som används i inläsningsprocessen. Istället kanske det skulle vara snabbare att hämta all data i en databasförfrågan och sedan dela upp och sortera det i programmet.

5.4 Python

Som vi har redovisat i resultatet har hastigheterna när man använder Python för inläsning mycket snabbare än Matlab eller C++. Python var dock det slternativ som använde mest minne under inläsningsprocessen. På grund utav att inläsningen i Python var så snabb skulle det vara intressant att undersöka om analysen även skulle vara snabbare än i Matlab. Om den är snabbare eller något långsammare så kan Python vara det bästa alternativet av de språk som testats för att vidareutveckla programmet. Om det inte är snabbare så kan Python användas för att hantera inläsningen tillsammans med något annat språk som hanterar analysen.

Eftersom FreeOpcUa även finns tillgängligt för Python så kan det även vara intressant att testa att skriva om OPC-klienten i Python då det verkar som att databaskopplingen är snabbare i Python.

5.5 Matlab

I nuläget finns det en server som har regulatordata, en klient som pratar med servern och hämtar data samt lägger in den i en databas. Matlab programmet som hämtar data från databasen och sedan utför sortering och analys.

Ett förslag som togs upp var om flödet hade blivit mer effektivt om klienten togs bort och Matlab hade hämtat regulatordata från servern direkt. Om Matlab används som en klient finns fortfarande kravet för att skicka hämtade data till en databas för att ha ett arkiv.

Matlab har ett bibliotek för att skapa en OPC koppling så att använda Matlab som klienten är i praktiken möjligt men det skulle kräva att vissa moment i koden skrivs om. I nuläget används index av regulatorerna i *Regulators*-tabellen i databasen för att komma ihåg vilka värden som hör till vilken regulator men om Matlab skulle fungera som klienten skulle därmed namnen på regulatorerna användas i stället för att hålla ordning på vilka värden som hör till vilken regulator.

Kontroll om vilka värden som hör till vilken regulator skulle också kunna ske på samma sätt som i nuläget genom att regulatorerna sparas i databasen via Matlab och sedan hämtas men det skulle vara ineffektivt att skicka data till databasen från Matlab för att sedan hämta samma data igen senare.

5.6 Minska slöseri av resurser

Förhoppningen är att detta arbete ska kunna bidra till att minska slöseri av resurser inom industrier. Under utvecklingen av programmet har fokus lagt på att programmet skall kräva så få processorcykler som möjligt för att datorn inte ska arbeta konstant med analysen, och därmed använda mindre ström.

I ett system med en dåligt trimmad regulator kan uppstå svängningar i styrsignalen som sedan kan orsaka att styrdonet slits ut på grund av ständiga förändringar. Tanken är att programmet ska kunna användas för att hitta dåligt presterande regulatorer så att mängden utrustning som behöver slängas eller bytas kan dras ner. I stället kan regulatorn trimmas när fel har uppkommit vilket minskar slöseri på material och sparar pengar.

6 Slutsats och framtida rekommendationer

Projektet har uppnått sitt mål om att fungera i realtid och göra en analys en gång i timmen. De olika programmen har gått igenom flera stadier där de har förbättrats, testats och förfinats. I nuläget fungerar OPC kopplingen inom kravet på en uppdatering i sekunden för upp till 2000 regulatorer. Databasen har ett arkiv med gamla uppdateringar ifall de skulle behövas samtidigt som analysen utförs på de uppdateringar som blir skickade varje sekund. Databasen kan också hantera ändringar av hur regulatorerna fungerar eller ser ut dynamiskt under körningen. Sist men inte minst har både hämtningen av data från databasen ändrats och gjorts ungefär 90 procent snabbare generellt och ~98 procent snabbare för 100 regulatorer.

Förbättringen beror på att färre tillstånd hämtas vid varje analys och att koden hämtar regulatordata från en databas. Analyskoden har också gåtts igenom och fel har hittats och rättats till så att analysen sker ~39 procent snabbare för 10 000 regulatorer och hastigheten för få regulatorer var aldrig något problem men även där är analysen ~15 procent snabbare.

Om C++ fortfarande är ett alternativ i framtida projekt så rekommenderar vi att undersöka om det finns ett snabbare alternativ att hämta data från databasen eller att låta Python göra detta.

Om fortsättningsarbeten görs kan ett av sätten att snabba upp inläsningen av data vara att skriva den delen av koden i Python och generera en .mat-fil som sedan kan analyseras. Det är också möjligt att skippa sparningen av data i en databas och i stället använda analysprogrammet som OPC-klient men då måste programmet även ta hand om arkiveringen av gamla uppdateringar.

Källförteckning

[1] R. Ernfjäll, J. Larson, “Diagnostisering av regulatorer”, Chalmers University of Technology, Göteborg, Sverige, 2017. [Online]. Tillgänglig: https://odr.chalmers.se/handle/20.500.12380/251259, Hämtad: 2021-03-20

[2] G. Rosin, P. Svensson, “Systemidentifiering och analys av olinjäriteter i styrdon”, Chalmers University of Technology, Göteborg, Sverige, 2018. [Online]. Tillgänglig: https://odr.chalmers.se/handle/20.500.12380/255421, Hämtad: 2021-03-20

[3] Oracle, “What Is a Database?”. [Online]. Tillgänglig: https://www.oracle.com/database/what-is-database/ (hämtad: 2021-05-16)

[4] Oracle, ”What Is a Relational Database?”. [Online]. Tillgänglig: https://www.oracle.com/database/what-is-a-relational-database/ (hämtad: 2021-05-16)

[5] OPC Foundation, “What is OPC?”. [Online]. Tillgänglig: https://opcfoundation.org/about/what-is-opc/ (hämtad: 2021-05-13)

[6] OPC Foundation, “Unified Architecture”. [Online]. Tillgänglig: https://opcfoundation.org/about/opc-technologies/opc-ua/ (hämtad: 2021-05-13

[7] IBM, “SOA (Service-Oriented Architecture)”. [Online]. Tillgänglig: https://www.ibm.com/cloud/learn/soa (hämtad: 2021-05-16)

[8] L. Prechelt, "An empirical comparison of seven programming languages", *Computer*, vol. 33, nr. 10, ss. 23-29, Okt. 2000, doi: 10.1109/2.876288.

[9] Standard C++, “Big Picture Issues”, 2021. [Online]. Tillgänglig: https://isocpp.org/wiki/faq/big-picture#what-is-cpp, (hämtad: 2021-05-13)

[10] University Information Technology Services, “What is the difference between a compiled and an interpreted program?”, 2018. [Online]. Tillgänglig: https://kb.iu.edu/d/agsz#:~:text=The%20difference%20between%20an%20interpreted,program%20written%20in%20assembly%20language. (hämtad: 2021-05-19)

[11] Microsoft, “About Vcpkg,” 2021. [Online]. Tillgänglig: https://vcpkg.io/en/getting-started.html (hämtad: 2021-05-13).

[12] LibXl, “Direct reading and writing Excel files”, 2021. [Online]. Tillgänglig: https://www.libxl.com/home.html (hämtad 2021-05-14)

[13] LibXl, “Download”, 2021. [Online]. Tillgänglig:https://www.libxl.com/download.html (hämtad 2021-05-14)

[14] Libpqxx, “The C++ connector for PostgreSQL,” 2021. [Online]. Tillgänglig: http://pqxx.org/development/libpqxx/ (hämtad: 2021-05-08).

[15] MathWorks, “Matlab,” 2021. [Online]. Tillgänglig: https://www.mathworks.com/products/matlab.html (hämtad: 2021-05-13)

[16] MathWorks, “Pricing and Licensing”. [Online]. Tillgänglig: https://se.mathworks.com/pricing-licensing.html?prodcode=ML&intendeduse=comm (hämtad: 2021-05-12)

[17] MathWorks, “Get and Manage Add-Ons”. [Online]. Tillgänglig: https://se.mathworks.com/help/matlab/matlab_env/get-add-ons.html (hämtad: 2021-05-19)

[18] Python, “General Python FAQ”. [Online]. Tillgänglig: https://docs.Python.org/3/faq/general.html#what-is-python (hämtad: 2021-05-19)

[19] NumPy, “NumPy”. [Online]. Tillgänglig: https://numpy.org/ (hämtad: 2021-05-19)

[20] Pandas, ”About pandas,” 2021. [Online]. Tillgänglig: https://pandas.pydata.org/about/ (hämtad: 2021-05-10).

[21] PyPI, “psycopg2”, 2020. [Online]. Tillgänglig: https://pypi.org/project/psycopg2/ (hämtad: 2021-05-19)

[22] The Psycopg Team, “Psycopg 2.8.7.dev0 documentation,” 2021. [Online]. Tillgänglig: https://www.psycopg.org/docs/ (hämtad: 2021-05-13).

[23] Microsoft, “Visual Studio 2019”, 2021. [Online]. Tillgänglig: https://visualstudio.microsoft.com/vs/ (hämtad: 2021-05-13)

[24] Microsoft, “Visual Studio Code”, 2021. [Online]. Tillgänglig: https://code.visualstudio.com/ (hämtad: 2021-05-13)

[25] FreeOpcUa, “FreeOpcUa: Open Source C++ and Python OPC-UA Server and Client Libraries and Tools”. [Online] Tillgänglig: http://freeopcua.github.io/ (hämtad: 2021-05-16)

[26] FreeopcUa, “Open Source C++ OPC-UA Server and Client Library”. [Online]. Tillgänglig: https://github.com/FreeOpcUa/freeopcua (hämtad: 2021-05-16)

[27] PostgreSQL, “38.1. Overview of Trigger Behavior”. [Online]. Tillgänglig: https://www.postgresql.org/docs/13/trigger-definition.html (hämtad: 2021-05-19)

[28] The PostgreSQL Global Development Group, “About”, 2021. [Online]. Tillgänglig: https://www.postgresql.org/about/ (hämtad: 2021-05-05)

[29] pgAdmin, “FAQ”, 2021. [Online]. Tillgänglig: https://www.pgadmin.org/faq/ (hämtad: 2021-05-13)

[30] PostgreSQL, “PostgreSQL 13.3 Documentation”. [Online]. Tillgänglig: https://www.postgresql.org/docs/13/index.html (hämtad: 2021-05-13)

[31] D. Galles, “Data Structure Visualizations,” myUSF, [Online]. Maj. 2021. Tillgänglig: https://www.cs.usfca.edu/~galles/visualization/Algorithms.html (hämtad: 2021-05-25)

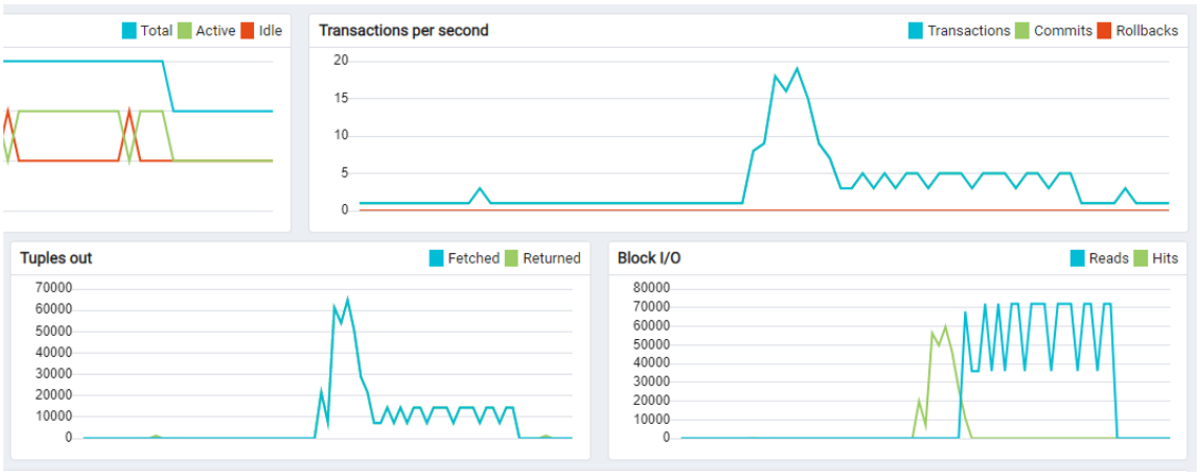
[32] California State Polytechnic University, ”CS241 -- Lecture Notes: B-Trees,” 2021. [Online]. Tillgänglig: https://www.cpp.edu/~ftang/courses/CS241/notes/b-tree.htm (hämtad: 2021-05-19)

[33] MathWorks, “PostgreSQL Native Interface,” 2021. [Online]. Tillgänglig: https://www.mathworks.com/help/database/postgresql-native-interface.html (hämtad: 2021-04-06)

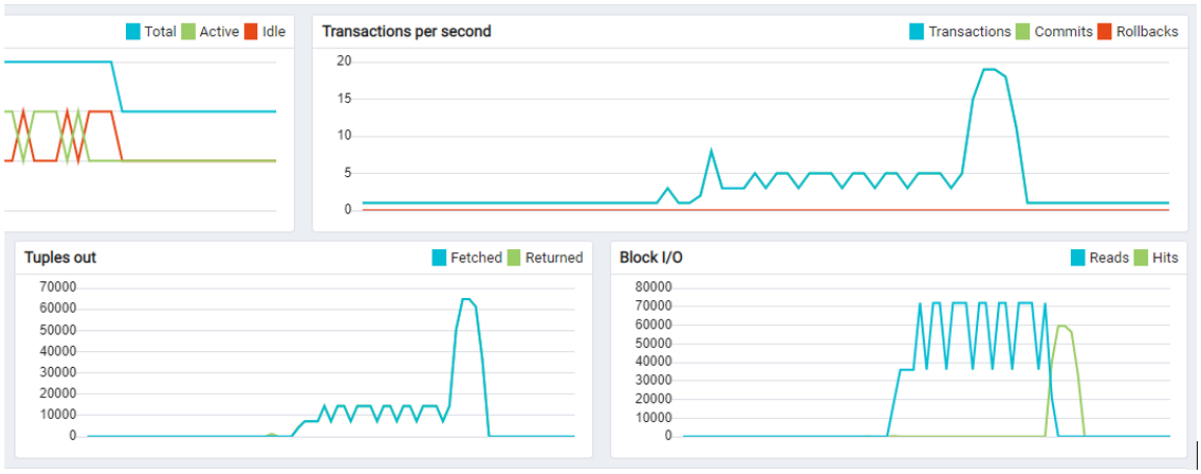
[34] JSON, “Introducing JSON”. [Online]. Tillgänglig: https://www.json.org/json-en.html (hämtad: 2021-05-20)

[35] Y. Shichkina, "Approaches to Speed up Data Processing in Relational Databases", Procedia Computer Science, vol. 150, s. 131-139, 2019, doi: 10.1016/j.procs.2019.02.026.

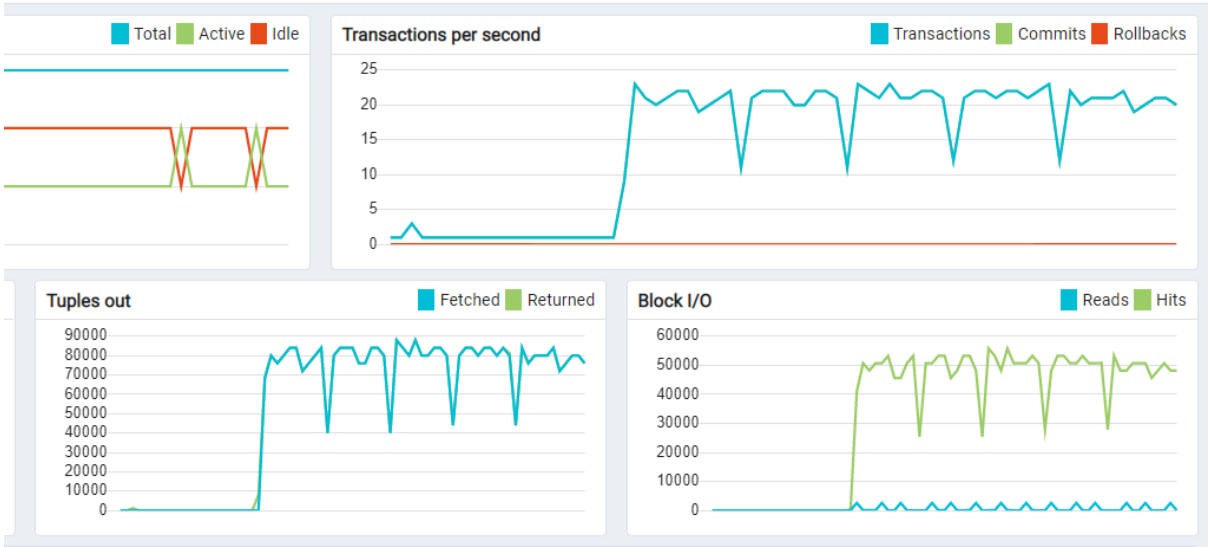
Bilagor



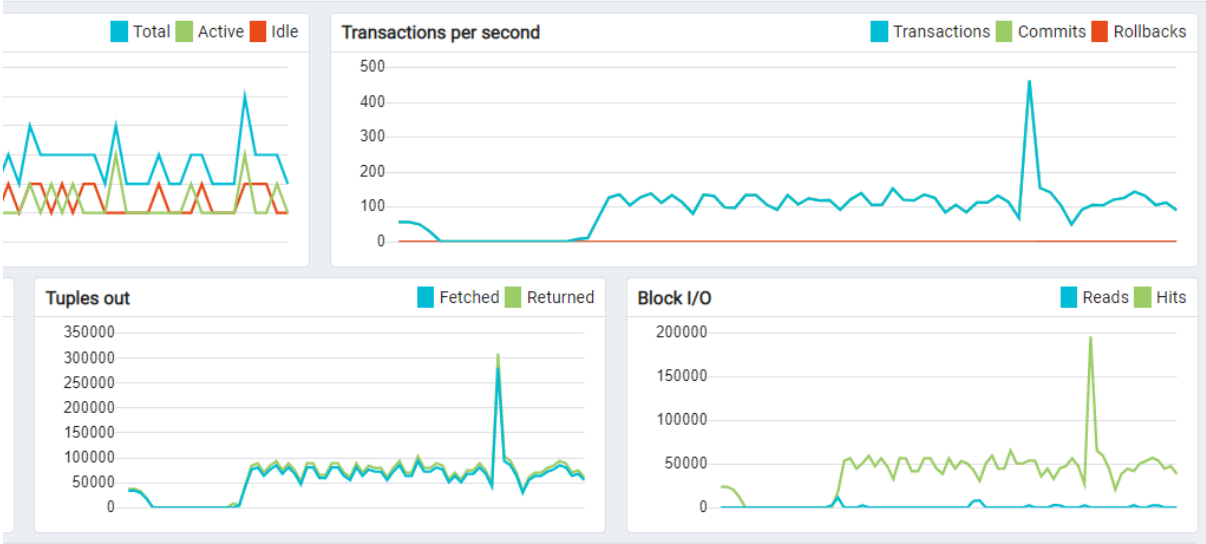
Bilaga 1: Hämtning i ordning A,B,C



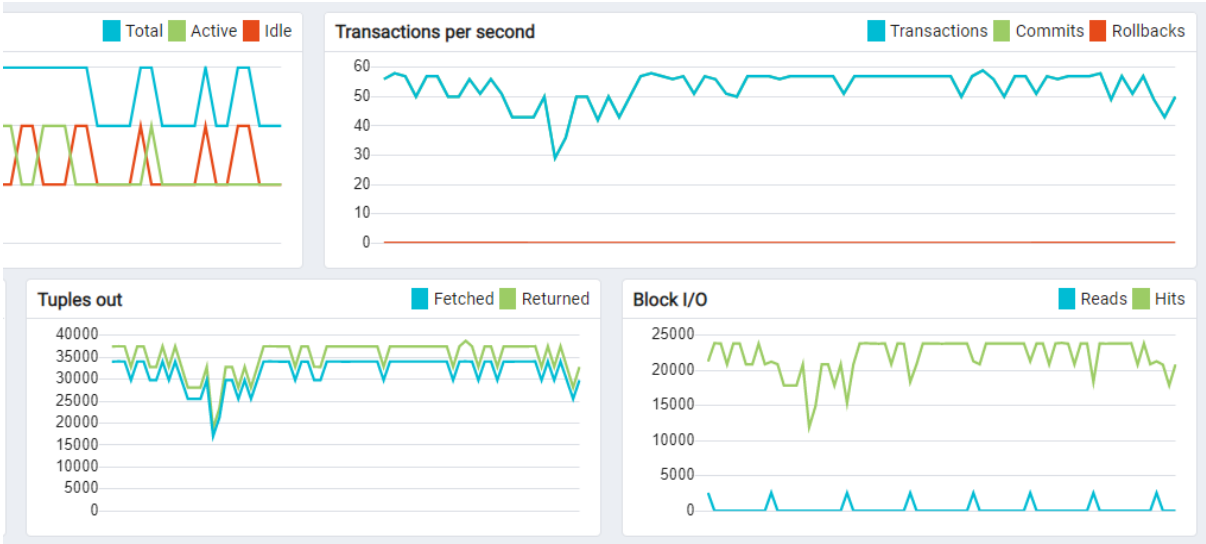
Bilaga 2: Hämtning i ordning C, B, A



Bilaga 3: Vanlig drift vid hämtning av regulatorer



Bilaga 4: Parfor med anlustning/stänging i loopen.



Bilaga 5: Här skedde ingen parallellisering men fler anslutningar gjordes detta drog upp antalet transaktioner per sekund men antalet tuples som skickades gick ner. Mer tid dras också i loopen eftersom fler rader kod behöver utföras i koden.

	id [PK] text	regtype text	sprangelow real	sprangehigh real	spunit text	pvrangelow real	pvrangehigh real	pvunit text	outrangelow real	outrangehigh real	outunit text	ctrlaction text	gain real	reset real	rate real
1	81FC10	FC	0	439	m3/h	0	439	m3/h	0	100	%	Reverse	0.1	240	0
2	81FC102	FC	0	121574	Sm³/h	0	121574	Sm³/h	0	100	%	Reverse	0.65	15	0
3	81FC106	FC	0	100	Sm³/h	0	100	Sm³/h	0	100	%	Reverse	0.3	15	0
4	81FC122	FC	0	31900	kg/h	0	31900	kg/h	0	100	%	Reverse	1.5	5	0
5	81FC124	FC	0	439	m3/h	0	439	m3/h	0	100	%	Reverse	0.03	3	0
6	81FC125	FC	0	439	m3/h	0	439	m3/h	0	100	%	Reverse	.0588888	6.05449	0
7	81FC128	FC	0	41	m3/h	0	41	m3/h	0	100	%	Reverse	0.1	10	0
8	81FC130	FC	0	59887	m3/h	0	59887	m3/h	0	100	%	Reverse	0.74	18	0
9	81FC14	FC	0	1140	Sm³/h	0	1140	Sm³/h	0	100	%	Reverse	0.07	5.3	0
10	81FC143	FC	0	2000	kg/h	0	2000	kg/h	0	100	%	Reverse	0.2	35	0
11	81FC24	FC	0	500	m3/h	0	500	m3/h	0	100	%	Reverse	0.29	28	0
12	81FC25	FC	0	301	m3/hr	0	301	m3/h	0	100	%	Reverse	0.5	8	0
13	81FC27	FC	0	130	m3/h	0	130	m3/h	0	100	%	Reverse	0.15	9.62	0
14	81FC29	FC	0	24055	Sm³/h	0	24055	Sm³/h	0	100	%	Reverse	0.1	7.22	0
15	81FC3	FC	0	364	m3/h	0	364	m3/h	0	100	%	Reverse	0.45	6.59	0
16	81FC39	FC	0	152	m3/h	0	152	m3/h	0	100	%	Reverse	0.04	23.58	0
17	81FC41	FC	0	250	m3/h	0	250	m3/h	0	100	%	Reverse	0.57	9.7	0
18	81FC48	FC	0	4500	kg/h	0	4500	kg/h	0	100	%	Reverse	0.1	25	0
19	81FC5	FC	0	450	m3/h	0	450	m3/h	0	100	%	Reverse	0.39	9	0
20	81FC51	FC	0	236	m3	0	236	m3	0	100	%	Reverse	0.06	15	0
21	81FC53	FC	0	450	m3/hr	0	450	m3/hr	0	100	%	Reverse	0.43	22.38	0
22	81FC59	FC	0	4950	kg/h	0	4950	kg/h	0	100	%	Reverse	0.15	25	0

Bilaga 6: Tabell av regulatorer i pgAdmin

	time [PK] timestamp without time zone	regulator [PK] text	sp real	pv real	error real	output real	mode real
1	2021-04-20 16:08:37.55793	test0	30	29.2737	0.7262993	10.2709	16
2	2021-04-20 16:08:37.55793	test1	215	347.487	-132.487	0	16
3	2021-04-20 16:08:37.55793	test10	3600	3596.72	3.2800293	61.5722	16
4	2021-04-20 16:08:37.55793	test11	193.547	195.335	-1.7880096	73.6055	32
5	2021-04-20 16:08:37.55793	test12	388.103	388.103	0	0	8
6	2021-04-20 16:08:37.55793	test13	3450	3444.95	5.050049	42.9442	16
7	2021-04-20 16:08:37.55793	test14	165.911	166.076	.16500854	64.3649	32
8	2021-04-20 16:08:37.55793	test15	650	650.777	.77697754	96.9287	16
9	2021-04-20 16:08:37.55793	test16	5.32409	5.26187	)62220097	30.4585	32
10	2021-04-20 16:08:37.55793	test17	82.8745	82.8745	0	78.71	8

Bilaga 7: Bild på tabellen *StatesArchive*

	id text	regtype text	sprangelow real	sprangehigh real	spunit text	pvrangelow real	pvrangehigh real	pvunit text	outrangelow real	outrangehigh real	outunit text	ctrlaction text	gain real	reset real	rate real
1	test0	FC	0	364	m3/h	0	364	m3/h	0	100	%	Reverse	0.45	6.59	0
2	test1	FC	0	450	m3/h	0	450	m3/h	0	100	%	Reverse	0.39	9	0
3	test2	FC	0	439	m3/h	0	439	m3/h	0	100	%	Reverse	0.1	240	0
4	test3	FC	0	1140	Sm³/h	0	1140	Sm³/h	0	100	%	Reverse	0.07	5.3	0
5	test4	FC	0	500	m3/h	0	500	m3/h	0	100	%	Reverse	0.29	28	0
6	test5	FC	0	301	m3/hr	0	301	m3/h	0	100	%	Reverse	0.5	8	0
7	test6	FC	0	130	m3/h	0	130	m3/h	0	100	%	Reverse	0.15	9.62	0
8	test7	FC	0	24055	Sm³/h	0	24055	Sm³/h	0	100	%	Reverse	0.1	7.22	0
9	test8	FC	0	152	m3/h	0	152	m3/h	0	100	%	Reverse	0.04	23.58	0
10	test9	FC	0	250	m3/h	0	250	m3/h	0	100	%	Reverse	0.57	9.7	0

Bilaga 8: *RegulatorsView*

	regulator text	sp real	pv real	output real	mode real
1	test0	30	29.2737	10.2709	16
2	test1	215	347.487	0	16
3	test2	343	342.34	72.1811	16
4	test3	690	689.204	60.5344	16
5	test4	307.961	308.268	30.5005	32
6	test5	-16.7884	-16.7884	0	8
7	test6	73.1243	73.5247	43.365	32
8	test7	0	0	0	8
9	test8	110	110.208	36.1752	16
10	test9	97.7784	115.95	0	16

Bilaga 9: *StatesViews* uppbyggnad i databasen

```
CREATE FUNCTION Insert_States() RETURNS trigger AS $$
   DECLARE error REAL;
   BEGIN
      IF(EXISTS (SELECT id FROM Regulators WHERE id = NEW.regulator)) THEN
         error := NEW.sp - NEW.pv;
         INSERT INTO States VALUES (CURRENT_TIMESTAMP, NEW.regulator, NEW.sp, NEW.pv, error,
NEW.output, NEW.mode);
         INSERT INTO StatesArchive VALUES (CURRENT_TIMESTAMP, NEW.regulator, NEW.sp, NEW.pv,
error, NEW.output, NEW.mode);
      END IF;
      RETURN NEW;
   END;
$$ LANGUAGE plpgsql;

CREATE FUNCTION Insert_Regulators() RETURNS trigger AS $$
   BEGIN
      IF(EXISTS (SELECT id FROM Regulators WHERE id = NEW.id)) THEN
         UPDATE Regulators SET regType = NEW.regType, spRangeLow = NEW.spRangeLow, spRangeHigh
= NEW.spRangeHigh,
            spUnit = NEW.spUnit, pvRangeLow = NEW.pvRangeLow, pvRangeHigh = NEW.pvRangeHigh,
pvUnit = NEW.pvUnit, outRangeLow = NEW.outRangeLow,
            outRangeHigh = NEW.outRangeHigh, outUnit = NEW.outUnit, ctrlAction = NEW.ctrlAction, gain =
NEW.gain, reset = NEW.reset, rate = NEW.rate
            WHERE id = NEW.id;
      ELSE
         INSERT INTO Regulators VALUES (NEW.id, NEW.regType, NEW.spRangeLow, NEW.spRangeHigh,
            NEW.spUnit, NEW.pvRangeLow, NEW.pvRangeHigh, NEW.pvUnit, NEW.outRangeLow,
            NEW.outRangeHigh, NEW.outUnit, NEW.ctrlAction, NEW.gain, NEW.reset, NEW.rate);
      END IF;
      RETURN NEW;
   END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER Insert_States INSTEAD OF INSERT ON StatesView
   FOR EACH ROW EXECUTE FUNCTION Insert_States();

CREATE TRIGGER Insert_Regulators INSTEAD OF INSERT ON RegulatorsView
   FOR EACH ROW EXECUTE FUNCTION Insert_Regulators();
```

Bilaga 10: SQL-kod för triggers och deras funktioner

```
-- Regulators(_name_, type, sprangelow, sprangehigh, spunit, pvrangelow, pvrangehigh, pvunit, outrangelow,
outrangehigh, outunit, ctrlaction, gain, reset, rate)
CREATE TABLE Regulators(
    id TEXT,
    regType TEXT NOT NULL,
    spRangeLow REAL,
    spRangeHigh REAL,
    spUnit TEXT,
    pvRangeLow REAL,
    pvRangeHigh REAL,
    pvUnit TEXT,
    outRangeLow REAL,
    outRangeHigh REAL,
    outUnit TEXT,
    ctrlAction TEXT,
    gain REAL,
    reset REAL,
    rate REAL,
    PRIMARY KEY(id)
);

-- States(_time_, _regulator_, sp, pv, output, mode)
CREATE TABLE States(
    time TIMESTAMP,
    regulator TEXT,
    sp REAL NOT NULL,
    pv REAL NOT NULL,
    error REAL NOT NULL,
    output REAL NOT NULL,
    mode REAL NOT NULL,
    PRIMARY KEY(time, regulator),
    FOREIGN KEY(regulator) REFERENCES Regulators(id)
);

-- StatesArchive(_time_, _regulator_, sp, pv, output, mode)
CREATE TABLE StatesArchive(
    time TIMESTAMP,
    regulator TEXT,
    sp REAL NOT NULL,
    pv REAL NOT NULL,
    error REAL NOT NULL,
    output REAL NOT NULL,
    mode REAL NOT NULL,
    PRIMARY KEY(time, regulator),
    FOREIGN KEY(regulator) REFERENCES Regulators(id)
);

CREATE VIEW RegulatorsView AS SELECT * FROM regulators;

CREATE VIEW StatesView AS SELECT regulator, sp, pv, output, mode FROM states;

--CREATE INDEX hash_state_reg ON public.states USING HASH (regulator);
CREATE INDEX b_tree_state_reg  ON public.states (regulator);
```

Bilaga 11: SQL-kod för tabeller, views och index