Løsninger når store datamengder skal legges inn i PostgreSQL

Eva Solbjørg Flo Heggem, Lars Opsahl[1] og Svein Solberg[2]

Eva Solbjørg Flo Heggem et al.: Solutions when big data are inserted into PostgreSQL

KART OG PLAN, Vol. 77, pp. 121–126, POB 5003, NO-1432 Ås, ISSN 0047-3278

The Norwegian Institute of Bioeconomy Research (NIBIO) manages and analyzes large amounts of data. In this article we describe how billions of weather model values were inserted into a manageable PostgreSQL database and linked to both spatial data and other tabular data The challenge was to find ways of organizing and managing the insertion of raw data into the database in a realistic time frame, so that the data could be queried and used in spatial analysis. The result shows how Open Source software can handle large amounts of data using simple scripts and without requiring a high performance computer server.

Key words: Open Source, PostgreSQL, PostGIS, GNU parallel, GIS

Eva Solbjørg Flo Heggem, Geomatikk NIBIO, Postboks 115, NO-1431 Ås,
epost: Eva.Heggem@nibio.no

Innledning

Trenger du å kjøpe Enterprise programvare og et stort databasekluster for å kunne legge inn 1 million rader i sekundet i en database eller gjøre spørringer mot 30 milliarder verdier som er tid- og stedfestet? Nei, det eneste du trenger er en PostgreSQL-database og enkle shell-skript. PostgreSQL og slike enkle skript brukes i dag over hele verden og er gratis å laste ned og bruke for alle.

En av kjerneoppgavene til NIBIO (Norsk institutt for bioøkonomi) er å etablere og drifte effektive, brukervennlige løsninger for informasjon om arealressursene, dvs. forvaltning, produksjon og formidling av digitale data. I avdeling for geomatikk satser vi på åpen og fri programvare og bruker PostgreSQL/PostGIS-databaser i forvaltningen av kartdata og databaser.

Selv om hoveddelen av kartdataene forvaltes gjennom disse løsningene tilkommer det store mengder stedfestede data som de ulike forskermiljøene bruker og som forvaltes i egne systemer. Skogforskningen bruker de største datamengdene i NIBIO. Der brukes det i stadig større grad store mengder laser- og bildedata fra droner, fly og satellitt. Dette er data med høy oppløsning som dekker store områder og gir detaljert kunnskap – noen ganger helt ned på enkelttrenivå. Dataene kan sammenstilles med informasjon om f.eks. bestandsdata og gir mange muligheter for utvikling av beregningsmodeller som gir økt kunnskap om skog og skogbruk.

Eksempler på andre viktige datasett i skogforskningen og for andre fagfelt er data om vær og klima. WISLINE (Wind, Ice and Snow Load Impacts on Infrastructure and the Natural Environment) er et stort, tverrfaglig prosjekt finansiert av Norges forskningsråd, og med deltakelse fra blant annet Meteorologisk institutt, klimaforskere og skogforskere ved NIBIO. Formålet i prosjektet er å se på vind, ising og snø og deres påvirkning på infrastruktur og miljø i Norge. For å lage modeller over skogens utvikling og prognoser må vi koble sammen disse dataene med andre data som terrengparametere og bestandsdata. Tilsvarende bruker de som forsker på jordsmonnets egenskaper vær- og klimadata i sine modeller, for å finne dyrkningspotensial i den dyrka og dyrkbare jorda i Norge. Dyrkbar jord er arealer som ved oppdyrking kan settes i stand slik at de vil

1. Lars Aksel Opsahl, Geomatikk NIBIO
2. Svein Solberg, Landsskogtakseringen NIBIO

holde kravene til fulldyrka jord og som holder kravene til klima og jordkvalitet for plantedyrking. Vær- og klimadata inneholder store datamengder og lange tidsserier. Det er derfor knyttet flere utfordringer til forvaltningen av dem slik at forskerne kan benytte seg av dem på en hensiktsmessig måte.

I denne artikkelen vil vi fokusere på utfordringer og løsninger knyttet til å integrere mange milliarder værdataverdier inn i våre databaser og gjøre dem tilgjengelige for analyser. Det er viktig for oss at metodikken er enkel, repeterbar og ikke mer avansert enn nødvendig. Oppgaven ble først løst for WISLINE-dataene og senere gjentatt for værdataserier for dyrkbar jord.

Metode

Før værdataene kan tas i bruk, må de legges inn i et hensiktsmessig forvaltningssystem. Vi velger å legge dem inn i PostgreSQL-databasen vår for enkelt å kunne trekke ut data fra selve datasettet samt kunne sammenstille dataene med kartdata.

Maskin- og programvare

Til å løse oppgavene omtalt i denne artikkelen bruker vi kun gratis, åpen og fritt tilgjengelig programvare (Open Source software). All programvare er «ut av boksen», det vil si at vi ikke har gjort egne endringer i kildekoden. Vi benytter oss av programmene R, GNU parallel (*Tange 2011)* og PostgreSQL/PostGIS versjon 9.3 (beregning), 9.5 (produksjon) og 9.6 (uttesting). PostgreSQL er et kraftig Open Source databaseprogram og PostGIS er en overbygning til denne som gjør det mulig å forvalte stedfestede data og utføre romlige spørringer på dem. Innlegging av de store datasettene gjøres via *batch*-skripting (enkle tekstfiler) kjørt i parallell. I tillegg til å være enkle gir *batch*-skriptene våre oss en god dokumentasjon av hva som er gjort, og det er lett å gjøre oppgaven flere ganger.

Andre mye brukte programmer hos oss for håndtering og visualisering av geografiske data, er Python, QGIS og GDAL/OGR. Disse programmene kan kjøres både i Windows og Linux og når dermed ut til en stadig økende andel av GIS-brukerne i verden. Disse programmene kommuniserer godt med PostgreSQL sammen med programmer som f.eks. R og det kommersielle S-Plus. Kartene i denne artikkelen er laget i QGIS. Se liste med lenker til aktuell programvare i slutten av artikkelen.

I dette prosjektet har vi brukt en Linuxserver med SSD-disker, 32 virtuelle CPUer (16 dual core) samt 264 GB minne. PostgreSQL/PostGIS kjører på en SSD-disk på 6 TB. De stadig økende datamengdene gjør at vi opplever at diskplass, og ikke maskinkraft, er kritisk for om vi får løst oppgavene våre slik vi ønsker det.

Organisering og nedlastning av rådata

WISLINE-dataene er nedskalerte og interpolerte værdata i et 2,5 km x 2,5 km rutenett for Sør-Norge med ca. 98.000 punkt. WISLINE-dataene inneholder fem parametere: temperatur, nedbør, vindhastighet og vindretning for hver tredje time, samt døgnverdier for markfuktighet. Disse variablene er modellert gjennom årene 2000-2015 (totalt 15 år) og gir ca. 22 milliarder forskjellige verdier vi må holde orden på og legge inn. I tillegg har vi avledet to nye parametere slik at det blir sju verdier per observasjon i databasen, som gir nesten 30 milliarder stedfestede verdier.

WISLINE-dataene fikk vi som filer per måned og parameter. Filene er tilgjengelige i det selvbeskrivende og maskinuavhengige NetCDF-formatet og lastes ned ved hjelp av et R-skript. Skriptet holder orden på hvilke filer og variabler som skal lastes ned og lagrer disse som rene tekstfiler der hver rad inneholder x-koordinat, y-koordinat, tid og utvalgte variable. Hver nedlastet fil inneholder data fra hele området for en måned, dvs. om lag 24 millioner rader. R-skriptet ble kjørt i parallell slik at filene kunne lastes ned raskere til vår lokale server.

For meteorologiske værdata på dyrkbar jord etableres et datasett med tidsserie over 30 år. Dataene er et uttrekk fra værdata interpolert til 1 km x 1 km rutenettet over Norge. Dette datasettet består av litt over 1,6 milliarder rader med 10 verdier per rad knyttet til en observasjon per dag, altså ca. 16 milliarder forskjellige verdier knyttet til

ca. 150.000 geografiske punkt. Til forskjell fra WISLINE-dataene kom dataene fra dyrkbar jord-prosjektet på store ZIP-filer og er sortert geografisk og ikke på tid.

Skriptene brukt til å lese inn WISLINE-data måtte tilpasses noe for at værdata for dyrkbarjord skulle lastes inn effektivt. Skripteksemplene for innlesing under er hentet fra værdatasettet for dyrkbar jord. Håndteringen i databasen er derimot likt for begge eksempeldatasettene våre.

Innlesing og oppdatering av rådata til PostgreSQL-tabeller

De nedlastede dataene var nå klare til å leses inn i PostgreSQL-databasen. I de første forsøkene på å lese inn værdataene i PostgreSQL-tabeller ble det fort klart at å lese inn alt ville kreve veldig mye tid. Det måtte gjøres noen grep for å få ned kjøretiden.

Metode nummer 1: Splitte opp og sortere inngangsfilene

Værdatafilene som lastes ned, er store og inneholder millioner av rader. Det gjør at de er tunge å arbeide med og at det brukes en del minne hver gang de åpnes for lesning. Værdatafilene for dyrkbar jord ble levert som åtte filer geografisk gruppert.

Inngangsfilene optimaliseres og splittes opp i mer håndterbare størrelser før de legges inn i PostgreSQL: vi fjerner alle doble blanke, sorterer dataene på tid og grupperer de i sett på 100.000 rader i hver fil. Her er et eksempel på et shell-skript for å splitte opp og sortere en fil. Denne benytter kommandoene *sed, sort, cat and split*:

```
find ../* ! -name 'data_ready' -type d -name '*' | while read filename ; \
do \
c1="cat $filename/* | sed 's/ */ /g' | sed 's/^ *//g' | sort -k2 -k3 -n | split --additional-
suffix $(basename $filename) -l 100000"
eval "$c1" 2>>../error_log
done
```

Nå har vi delt opp de åtte inngangsfilene i 16.000 små filer sortert på tid og med 100.000 rader i hver fil.

Metode nummer 2: Etablere partisjonerte tabeller i PostgreSQL.

PostgreSQL takler svært store datamengder. Likevel, når tabellene blir veldig store, blir de tunge å håndtere. Vi valgte derfor å legge værdataene inn i en partisjonert tabell. I PostgreSQL fungerer en partisjonert tabell på samme måte som en vanlig tabell. En master-tabell holder styr på alle delene og alle innlegginger og spørringer peker direkte mot mastertabellen. Undertabellene bør da deles inn på en logisk og hensiktsmessig måte. Vi splitter dataene på år og måned. Klimadata for dyrkbar jord ble fordelt på nesten 450 undertabeller for en 30 årsperiode. Når tabellene er splittet, er de mye mer håndterlige, og PostgreSQL kan jobbe raskt i parallell mot dem. I praksis inneholdt ingen av våre sorterte inngangsfiler data fra mer enn to ulike måneder. Dermed vil hver innleggingsjobb kun jobbe på to tabeller om gangen. Denne tilnærmingen passer bra med hvordan PostgreSQL har implementert sin 'Multi Version Concurrency Control', dvs. håndtering av flere hendelser mot samme tabell.

Metode nummer 3: Legge inn data med GNU parallel

De splittede dataene er så klare til å legges inn i den partisjonerte databasetabellen. En enkel måte å gjøre dette på er å lage en batch-fil med en kommando per linje og kjøre dette med GNU parallell. Hver linje inneholder da en *copy* kommando som legger data inn i en midlertidig «*unlogged table*» i databasen. En ulogget tabell er betydelig raskere enn vanlige tabeller fordi transaksjonene da ikke logges.

Før de ca. 16.000 filene leses inn i databasen genereres en kommandofil for GNU parallel, slik at vi kan gjøre innleggingen i parallelle prosesser.

```
ls -t -d -1 $PWD/* | while read filename ; \
do \
echo "time psql sl -c\"select schema.add_part_fnc('"$filename"');\" 2>> /../mylog.log; " >>
/tmp/running.txt
done
```

Disse linjene overlager en kommandofil, som vist under, med ca 16.000 linjer, en linje for hver fil. Hver linje kaller PostgreSQL med *psql* og sender inn en av filene sammen med kall til en funksjon vi har laget for formålet:

```
time psql sl -c"select schema.add_part_fnc ('/../xzcrevest');" 2>> /../mylog.log;
time psql sl -c"select schema.add_part_fnc ('/../xzcrdvest');" 2>> /../mylog.log;
time psql sl -c"select schema.add_part_fnc ('/../xzcrcvest');" 2>> /../mylog.log;
time psql sl -c"select schema.add_part_fnc ('/../xzcrbvest');" 2>> /../mylog.log;
```

Vi har da et inngangsdatasett som består av ca. 16.000 like store PostgreSQL-jobber samlet i en kommandofil. Ved å kjøre kommandoen under kalles GNU parallel med 20 tråder.

```
time parallel --gnu -j 20 :::: /tmp/running.txt
```

PostgreSQL jobber da parallelt på et sett av 20 filer om gangen som hver arbeider uavhengig av hverandre mot databasen. For oppdateringer med nye data i databasen gjør vi det på samme måten, vi sorter inngangsdataene på tid og gjør oppdateringene i parallell.

Ved å bruke oppdelte og sorterte filer, partisjonerte tabeller og kontrollerte løkker samt GNU parallell klarte vi både å legge inn og oppdatere data med en hastighet på 1 million rader i sekundet. I de første forsøkene oppnådde vi bare 10.000 rader i sekundet. Kjøretiden er slik redusert fra 60 timer til 35 minutter for et datasett med 650 millioner rader. Med denne hastigheten er det uproblematisk å etablere store, romlige datasett i en vanlig PostgreSQL-database.

Analyser på dataene

Når databasene med alle værdataene er etablerte kan den brukes i analyser, beregninger og visualiseringer. WISLINE-datasettet inneholder nå 30 milliarder dataverdier fordelt på 98.000 geografiske datapunkter i Sør-Norge. Selv om det er store datasett lar det seg fint gjøre å presentere dem i kart. PostgreSQL/PostGIS kan enkelt kobles opp mot andre programvarer. Dataene kan visualiseres ved f.eks. Open Source programmet QGIS. Se figur 1 og 2 for eksempler på bruk av QGIS mot WISLINE-dataene.

For beregninger og spørringer i databasen er det mange muligheter for optimalisering. Vi går ikke videre i detalj på dette her. Et generelt triks er å kjøre spørringer mot databasen med GNU parallell for å redusere beregningstid. Organiseringen av databasen i partisjonerte tabeller gjør den mer fleksibel, og vi kan jobbe direkte på dataene uavhengig av minnekrevende indekser.

Oppsummering og erfaring

For oss har det vært svært nyttig å splitte data opp i små enheter. Når data brytes opp og kjøres i mindre filer trenger vi ikke en spesielt rask maskin eller mye minne. Det som trengs er plass til å lagre store datamengder. En annen fordel med å splitte opp kjøringene er at det har vært lettere å finne feil underveis, både i data og metodikk.

WISLINE-dataene ble levert i filer sortert på tid. Når værdata for dyrkbar jord-prosjektet skulle legges inn, var ikke dette tilfelle, og innlegningene våre gikk igjen sakte. Etter at dataene ble sortert på tid før de ble splittet i mindre filer, var innleggingen av dataene levert for geografiske områder, like rask som data levert forhåndssortert på tid. Dette vi-

ser at enkle grep som gjøres i små shellskript kan være avgjørende for kjøretid – ikke hvor kraftig databasen eller serveren er.

Tabellhåndtering og tabellstruktur har også vært viktig for å få dataene til å fungere godt. Ved å ta i bruk partisjonering av tabellene i PostgreSQL gikk kjøretiden igjen kraftig ned. Partisjonering av tabellene har også flere fordeler utover å gjøre innlegging, oppdatering og søk raskere. Rensk av tabeller (*Vacum full*) vil kreve stor diskplass og man kan risikere at store tabeller ikke kan behandles når det ikke er plass på disken til en kopi (*full dump*) av tabellen.

Vår erfaringer er at PostgreSQL håndterer parallelle prosesser veldig godt. Vi opplevde aldri låsing i tabellene våre selv om flere prosesser skriver mot samme tabell. GNU parallel er et verktøy som gjør det enkelt å kjøre kontrollerte prosesser i mange tråder. I PostgreSQL 9.6 finnes også innebygget parallellitet. Det virker lovende, men vi har ikke testet denne funksjonaliteten selv ennå.

Konklusjon

I våre siste innlesninger og oppdateringer til PostgreSQL-databasen er hastigheten nå oppe i 1 million rader i sekundet med 20 tråder. Når vi går fra 10.000 til 1 million rader i sekundet, så går vi fra 60 timer til 35 minutter for å oppdatere 650 millioner rader. Dette gjør at det blir enklere for forskere å jobbe mer kontinuerlig med testing og oppdatering av datasett.

Gjennom dette arbeidet har vi vist at store datamengder kan leses inn i en PostgreSQL-database ved hjelp av Open Source programvare uten at det kreves spesielt kraftig hardware eller dyre Enterprise programvarer. Begrensingen ser i praksis ut til å ligge i tilgjengelig lagringsplass. Store datamengder krever stor diskplass, og ved behov for sikkerhetskopiering (*backup* og *restore*) av databasene og tabellene må denne plassen dobles. F.eks bruker klimadata for jordsmonn over 225 GB plass i PostgreSQL-databasen fordelt på 450 undertabeller.

Det er viktig at forskere og dataeiere beholder kontroll, eierskap og oversikt over dataene etter at de er etablert i forvaltningssystemet. Kompetanse, kommunikasjon og opplæring må prioriteres for at løsningene skal tas i bruk i praksis. Tett samarbeid mellom ulike faggrupper er nødvendig og fruktbart og Open Source programvare har vist seg å kommunisere godt med ulike systemer og løsninger.

Denne artikkelen er ikke en systematisk, grundig testing av kjøretider og hastigheter. Skriptene våre er ikke optimalisert så de kan fortsatt gjøres raskere og sikrere. De er ment som eksempler på at det er mulig å behandle store datamengder på overkommelig tid på en Linux-server med PostgreSQL.

Lenker til aktuell programvare

QGIS: QGIS Development Team, 2017. QGIS Geographic Information System. Open Source Geospatial Foundation. URL http://qgis.osgeo.org

PostgreSQL: https://www.PostgreSQLql.org/

PostgreSQL Concurrency Control: https://www.postgresql.org/docs/current/static/mvcc.html

GNU-parallel: https://www.gnu.org/software/ parallel/

R: https://www.r-project.org/

NetCDF: http://www.unidata.ucar.edu/software/ netcdf/

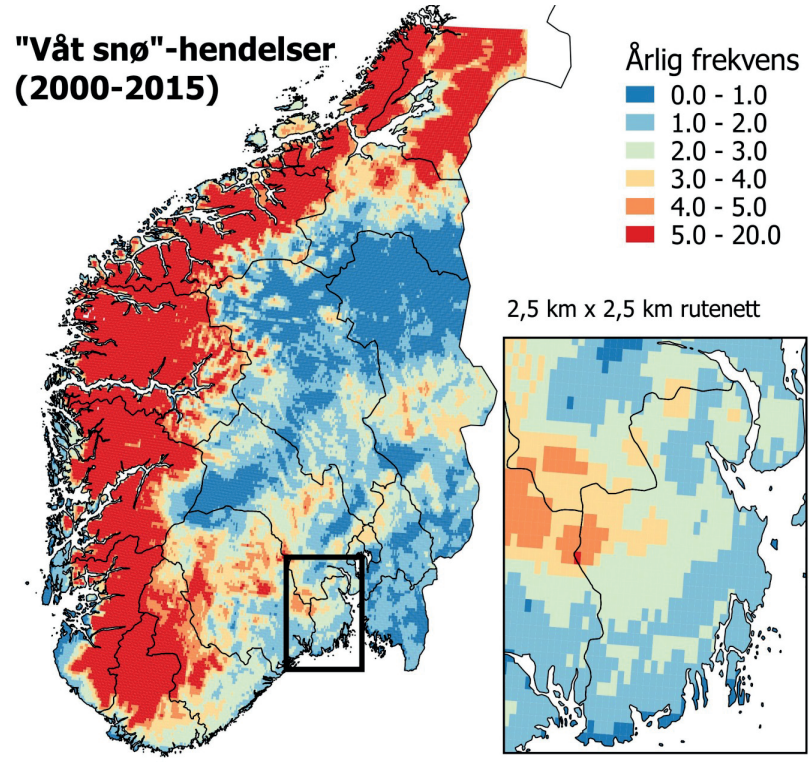
GDAL: http://www.gdal.org/

Python: https://www.python.org/

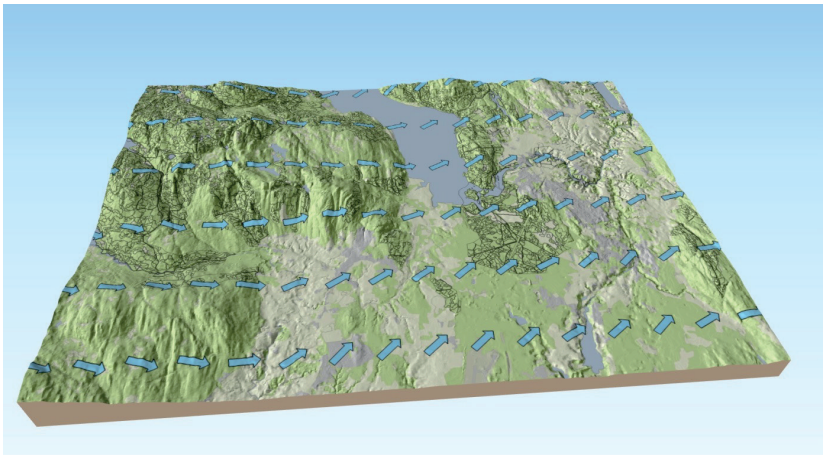
Shell Scripting: https://www.shellscript.sh/

Referanser

Tange (2011): GNU Parallel – The Command-Line Power Tool, ;login: The USENIX Magazine, February 2011:42-47.



Figur 1: Eksempel på visualisering av WISLINE-data rett fra PostgreSQL-databasen ved hjelp av QGIS. Figuren viser risiko for 'våt snø' hendelser i Sør-Norge. 'Våt snø' hendelser er døgn der nedbøren er mer enn 10 mm temperaturen er mellom 0,5 og 2 grader. Verdiene avledet fra WISLINE-parameterne nedbør og temperatur.



Figur 2: Eksempel på visualisering av WISLINE-data rett fra PostgreSQL-databasen ved hjelp av QGIS. Denne figuren viser vindretningen ved maksimal vindstyrke under stormen «Dagmar» desember 2011. PostgreSQL-tabellen er drapert over en 10 m terrengmodell og visualisert sammen med AR50-data og bestandsdata for skog. Qgis2threejs, en QGIS Python Plugin, er brukt for å få 3D-effekt.