

Resurssien optimointisimulaattorin toteutussuunnitelma

Elias Ervamaa

18. tammikuuta 2026

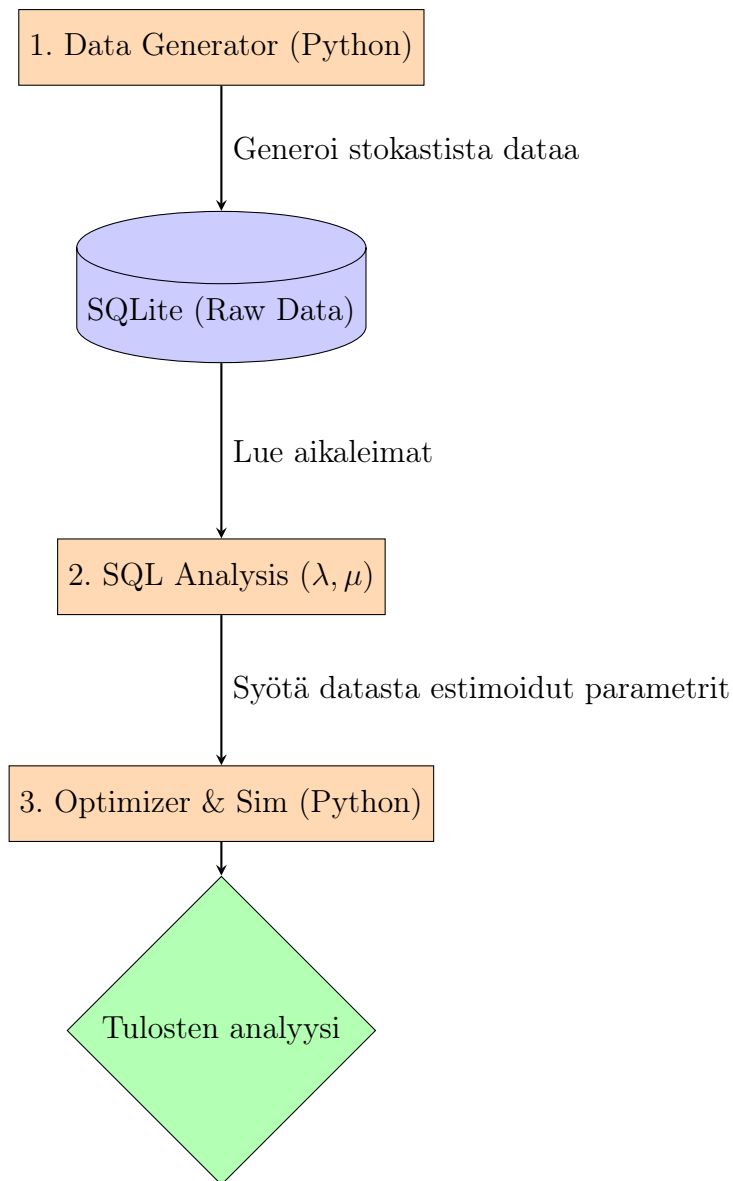
1 Johdanto

Tämä dokumentti kuvaa teknisen arkkitehtuurin ja toteutuksen vakuutusyhtiön resurssien allokointiongelman ratkaisemiseksi. Tavoitteena on validoida matemaattisesti johdettu optimointimalli käyttämällä data-vetoista lähestymistapaa.

Toteutus ei nojaa ennalta annettuihin parametreihin, vaan simuloi todellista Data Science -prosessia, jossa mallin syötteet on ensin louhittava raakadatasta SQL-kyselyillä.

2 Järjestelmäarkkitehtuuri

Ratkaisu koostuu kolmesta itsenäisestä moduulista, jotka muodostavat jalostusputken.



2.1 Moduulien kuvaus

1. Generoi synteettistä vakuutusdataa piilotetuilla parametreilla. Tämä simuloi reaaliaikailman ilmiötä, jonka lainalaisuudet ovat tuntemattomia.
2. Toimii analyytikon roolissa. Lukee raakadatan tietokannasta ja estimoi saapumis- ja palveluprosessien parametrit (λ, μ) SQL-aggregaatioilla.
3. Laskee teoreettisen optimin estimoitujen parametrien perusteella ja suorittaa diskreettitapahtumasimulaation (DES) verifioidakseen tuloksen.

3 Tietokantasuunnittelu

Datavarastona käytetään SQLite-relaatiotietokantaa. Skeema on suunniteltu tallentamaan tapahtumatasen dataa, josta prosessimittarit voidaan johtaa.

Taulu vakuutustapahtumat:

```
1 CREATE TABLE vakuutustapahtumat (  
2   id INTEGER PRIMARY KEY AUTOINCREMENT,  
3   vahinkotyyppi TEXT,      -- 'henkilo' tai 'auto'  
4   saapumisaika REAL,      -- Aikaleima (t)  
5   kasittely_alkoi REAL,   -- Aikaleima (t)  
6   kasittely_paattyi REAL -- Aikaleima (t)  
7 );
```

4 Parametrien estimointi

4.1 Saapumisintensiteetti (λ)

Poisson-prosessin intensiteetti lasketaan jakamalla havaintojen määrä havaintoajan pituudella.

```
1 --Lasketaan lambda (tapahtumaa per aikayksikko)  
2 SELECT  
3   vahinkotyyppi,  
4   COUNT(*) * 1.0 / (MAX(saapumisaika) - MIN(saapumisaika)) as  
   lambda_est  
5 FROM vakuutustapahtumat  
6 GROUP BY vahinkotyyppi;
```

4.2 Palvelunopeus (μ)

Palvelunopeus on keskimääräisen palveluajan käänteisluku ($\mu = 1/E[S]$). SQL:ssä las-
kemme ensin palvelun keston erotuksena paattyi - alkoi.

```
1 -- Lasketaan mu (palvelunopeus)  
2 SELECT  
3   vahinkotyyppi,  
4   1.0 / AVG(kasittely_paattyi - kasittely_alkoi) as mu_est  
5 FROM vakuutustapahtumat  
6 GROUP BY vahinkotyyppi;
```

5 Diskreettitapahtumasimulaatio

Simulaattori on toteutettu Pythonilla ilman ulkoisia kirjastoja.

Simulaatio noudattaa johdetun matemaattisen mallin oletusta (Skaalattu M/M/1), jossa resurssimäärä x mallinnetaan palvelukapasiteetin kasvuna. Systemi toimii kuten yksi "superpalvelija", jonka työskentelynopeus on x -kertainen yksittäiseen työntekijään verrattuna.

5.1 Toimintaperiaate

Simulaatio etenee tapahtumapohjaisesti. Aikajana ei ole jatkuva, vaan simulaatiokello hyppää aina suoraan seuraavan tapahtuman aikaleimaan.

Järjestelmän tilaa seurataan seuraavilla komponenteilla:

- **Tapahtumakalenteri:** Aikajärjestyksessä oleva lista tulevista tapahtumista. Tyy-
pit ovat *SAAPUMINEN* ja *POISTUMINEN*.
- **Tilamuuttujat:**
 - *Palvelijan tila:* Vapaa (0) tai Varattu (1).
 - *Jonon pituus:* $L(t)$.
- **Suorituskykymittarit:** Muuttujat, joihin kerätään dataa keskiarvojen laskemista
varten (esim. *kumulatiivinen odotusaika* ja *palveltujen asiakkaiden lkm*).

5.2 Algoritmi

Simulaatiosilmukka toimii seuraavalla logiikalla, kunnes asetettu simulointiaika T_{max} täyt-
tyy. Olkoon x resurssien määrä (kokonaisluku).

1. Asetetaan kello $t = 0$ ja luodaan ensimmäinen SAAPUMINEN-tapahtuma kalente-
riin.
2. **Pääsilmukka:**
 - (a) **Kellon siirto:** Otetaan kalenterista tapahtuma, jolla on pienin aikaleima, ja
siirretään kello t tähän aikaan.
 - (b) **Jos tapahtuma on SAAPUMINEN:**
 - Luodaan seuraava saapumisaika (Poisson-prosessi, intensiteetti λ) ja lisä-
tään se kalenteriin.
 - Jos palvelija on vapaa:
 - Asiakas pääsee palveluun.
 - Arvotaan palveluaika S . Huom: Koska käytössä on x resurssia, palve-
lunopeus on $x \cdot \mu$. Tällöin $S \sim \text{Exp}(x\mu)$.
 - Luodaan kalenteriin tapahtuma *POISTUMINEN* ajalle $t + S$.
 - Asetetaan palvelija varatuksi.
 - Jos palvelija on varattu:
 - Lisätään asiakas odotusjonoon.
 - (c) **Jos tapahtuma on POISTUMINEN:**

- Asiakas poistuu järjestelmästä.
- Jos odotusjonossa on asiakkaita:
 - Otetaan seuraava asiakas jonosta.
 - Arvotaan palveluaika $S \sim \text{Exp}(x\mu)$.
 - Luodaan kalenteriin uusi *POISTUMINEN* ajalle $t + S$.
 - Palvelija pysyy varattuna.
- Jos jono on tyhjä:
 - Asetetaan palvelija vapaaksi.

5.3 Optimointihaku (Grid Search)

Simulaattori ajaa yllä kuvattua algoritmia iteratiivisesti. Koska teoreettinen ratkaisu x^* on jatkuva luku, simulaattori testaa x^* :n ympärillä olevat kokonaislukuparit (x_1, x_2) , jotka toteuttavat budjettirajoitteen. Näin löydetään paras mahdollinen diskreetti allokaatio.