Jussi Loppukaarre

Serialisointiprosessin skaalautuvuus ja suorituskyky Apex-ohjelmointikielellä

Tietotekniikan Pro gradu -tutkielma

3. helmikuuta 2024

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Jussi Loppukaarre

Yhteystiedot: jussi.loppukaarre@gmail.com

Ohjaaja: Ville Isomöttönen

Työn nimi: Serialisointiprosessin skaalautuvuus ja suorituskyky Apex-ohjelmointikielellä

Title in English: Serialization process scalability and performance with Apex programming language

Työ: Pro gradu -tutkielma

Opintosuunta: Ohjelmisto- ja tietoliikennetekniikan opintosuunta

Sivumäärä: 58+13

Tiivistelmä: Tutkielma tarkasteli Salesforcen Apex-ohjelmointikielellä toteutettuja serialisointiprosesseja keskittyen erityisesti kääreluokan ja avain-arvo-pari-listan käyttöön. Tutkimuksen tavoitteena oli luoda yleistävät artefaktit ja arvioida näiden slaalautuvuutta tietuemäärän kasvaessa ja analysoida suorituskyvyllisiä eroja synkronisten ja asynkronisten suoritustapojen välillä.

Tutkielmassa hyödynnettiin suunnittelutieteen periaatteita, joita soveltamalla luotiin kaksi yleistävää artefaktia. Näiden artefaktien suorituskykyä mitattiin kokonaissuoritusaikana (ms), CPU-aikana (ms) ja käytetyn kekomuistin määränä (kt).

Tulosten perusteella havaittiin selkeitä suorituskyvyllisiä eroja kääreluokka ja avain-arvo-pari-lista -artefaktien välillä, mutta skaalautuvuus artefaktien välillä toteutui samankaltaisesti. Lisäksi tunnistettiin suorituskyvyllisiä eroja asynkronisten ja synkronisten suoritustapojen välillä.

Tutkielma tarjoaa tietoa Salesforcen Apex-ohjelmointikielellä toimiville ohjelmistokehittäjille, jotka pyrkivät optimoimaan suorituskykyä serialisointiprosesseissa. Tulokset avaavat myös uusia mahdollisuuksia jatkotutkimukselle, erityisesti laajempien prosessien kontekstissa.

Avainsanat: JSON, Serialisointi, Salesforce, Apex, Synkroninen, Asynkroninen, Skaalautuvuus, Suorituskyky

Abstract: The thesis examined serialization processes implemented in Salesforce's Apex programming language, with a particular focus on the use of wrapper class and key-value-pair-list. The objective of the study was to create artefacts and assess the scalability as the number of records increased and to analyze performance differences between synchronous and asynchronous execution methods.

The thesis applied principles of design science, resulting in the creation of two generalizing artifacts. The performance of these artifacts was measured in terms of total execution time (ms), CPU time (ms), and the amount of heap memory used (kb).

Based on the results, clear performance differences were observed between the wrapper class and key-value pair list, although scalability between the artifacts was similar. Additionally, performance disparities were identified between synchronous and asynchronous execution methods.

The thesis provides valuable insights for software developers working with Salesforce's Apex programming language, aiming to optimize performance in serialization processes. The findings also open up new avenues for further research, particularly in the context of broader processes.

Keywords: JSON, Serialization, Salesforce, Apex, Synchronous, Asynchronous, Scalability, Performance

Termiluettelo

Apex	Salesforce alustan oma ohjelmointikieli (Mathew ja Spraetz 2009).
Salesforce	Salesforce on SaaS-pilvipalvelu (Mathew ja Spraetz 2009).
Serialisointi	Objektin muuntaminen bittivirraksi tai tekstiformaattiin (Hericko ym. 2003; Hamerski ym. 2018).
SObjekti	Objekti joka voidaan tallentaa Lightning alusta tietokantaan (Salesforce 2023i).
Multitenant	Ohjelmistoarkkitehtuuri, jossa resurssit ovat jaettuna usean eri organisaation kesken (Walraven ym. 2012).

Kuviot

Taulukot

Sisällys

1 Johdanto

Erilaisten pilvipalvelujen hyödyntäminen liiketoiminnan harjoittamisessa on yleistä nykypäivänä. AlJahdali ym. (2014, s. 344) mukaan, pilvipalvelupohjaisten alustojen käyttäminen tasapainottaa joustavuutta ja kustannuksia. Eri tahot tarjoavatkin erilaisia pilvipalvelualustaratkaisuja. Yksi alustaratkaisu on palvelupohjaisten sovellusten periaate (multitenant), jossa eri käyttäjät voivat suorittaa omaa palveluaan samassa sovellusinstanssissa (Krebs, Momm ja Kounev 2012). Yksi tunnetuimmista pilvipalveluista, joka on rakennettu multitenant-periaatteella on Salesforce (Salesforce 2022b).

Multitenant-palvelut jakavat laitteistojen ja ohjelmistojen käyttöresursseja usean organisaation kesken, mutta suorituskyvyn eriyttämien organisaatioiden kesken voi olla haastava toteuttaa (Walraven ym. 2012). Tämän takia multitenant palvelu saattaa asettaa rajoitteita yhdelle transaktiolle eli suorituskerralle. Salesforce palvelu asettaa tiettyjä rajoituksia Apex-suorituksille, kuten tietokantakyselyjen määrä, käytössä olevan kekomuistin määrä ja transaktion maksimaalinen suoritusaika (Salesforce 2022a, Viitattu 21.10.2023). Ohelmistokehittäjien on täten otettava huomioon nämä rajoitteet kehittäessä Apex-ohjelmistoa.

Serialisointi on hyvin pitkään ja monipuolisesti tutkittu prosessi. Serialisointi prosessina tarkoittaa objektin muuntamista bittivirraksi (Hericko ym. 2003, s.44) tai ihmisen ymmärrettäviin tekstiformaatteihin, kuten XML tai JSON (Hamerski ym. 2018, s.773). Tässä tutkielmassa serialisointi on rajattu objektin muuntamiseen JSON tekstiformaattiin.

Tämän tutkielman tarkoituksena on keskittyä Salesforce alustan Apex-ohjelmointikielellä toteutettujen serialisointiprosessien skaalautuvuuden ja suorituskyvyn evaluointiin. Tutkielmassa on pyrkimyksenä vastata seuraaviin kysymyksiin:

1. Kuinka serialisointiprosessit kääreluokalla ja lista avain-arvo-pareilla skaalautuvat tietuemäärän kasvaessa?
2. Mitä suorituskyvyllisiä eroja on eri synkronisen ja asynkronisten Apex suoritustapojen välillä yhdessä edellä mainituilla serialisointiprosesseilla?

Tutkielman tutkimusmenetelmäksi on valittu suunnittelutiede (design science). Kyseistä me-

netelmään sovelletaan tutkimusongelmaan, koska sen avulla on mahdollista toteuttaa järjestelmällinen ja tavoitteellinen skaalautuvuuden ja suorituskyvyn analysointi. Suunnittelutieteessä korostetaan uusien ratkaisujen luomista ja niiden käytännön soveltamista (Hevner ym. 2004). Tämän menetelmän avulla oli mahdollista määrittää selkeät tavoitteet ja suunnitelmat tutkimuksen suorittamiseksi, sekä arvioida kehitettyjen artefaktien käyttökelpoisuutta ja tehokkuutta. Näin tutkimus voi tuottaa konkreettisia ja käytännön hyötyjä ohjelmistokehityksen ja suorituskyvyn optimoinnin saralla.

Tutkimuksen luvussa 2 on tarkoituksena avata Salesforcea ja serialisointia aiemmin toteutettujen tutkimuksien pohjalta. Luvussa 3 käsitellään tutkielmassa hyödynnettyä tutkimusmenetelmää ja esitetään toteutunut tutkimusprosessi. Luku 4 käsittää tutkimusprosessissa luodun aineiston analysoinnin. Luvussa 5 suoritetaan pohdinta tuloksista ja tutkimusprossessia. Viimeisessä luvussa esitetään johtopäätökset.

2 Katsaus Salesforceen ja serialisointiin

Tämän luvun tarkoituksena on esitellä perustavanlaatuinen kuvaus Salesforce-palvelun olemuksesta ja serialisoinnista. Lisäksi luku tarjoaa katsauksen siihen, kuinka edellä mainittua palvelua on tutkittu ja millaisia näkökulmia aiheeseen on aikaisemmissa tutkimuksissa esitetty. Luvussa esitettään myös tutkimusongelma.

2.1 Salesforce

Salesforce on SaaS-palvelu (Software-as-a-Service) (Mathew ja Spraetz 2009, s. 317). Kyseinen pilvipalvelu keskittyy asiakkuuksien-, myynnin- ja markkinoinhallintaan (Farrar 2023). Salesforcen vahvuutena on se, että käyttäjän ei tarvitse asentaa ohjelmistoja, vaan vain internetyhteys ja selain tarvitaan palvelun käyttöön (Manchar ja Chouhan 2017). Salesforce on ollut laaja-alainen tutkimuskohde. Tämän tutkielman keskeisenä tavoitteena on edistää Salesforceen liittyvää lisätutkimusta, jotta voimme saavuttaa alustalla tapahtuvien serialisointiprosessien käytännöistä alustavan yhteisymmärryksen.

Salesforcessa suoritettava dataprosessointi on ollut tuore tutkimuskohde. Marańda ym. (2022, s. 6-17) tutkivat asynkronisilla ja syknorisella Apex-metodeilla suoritettavan dataprosessoinnin suorituskykyä yhdessä MVC-arkkitehtuurin (model-view-controller architehcture) kanssa. Tuloksien perusteella synkroninen ja asynkroninen future-metodi eivät sovellu suurten tietuemäärien prosessointiin. Sen sijaan batch- ja queueable Apex-metodit soveltuivat suurten tietuemäärien dataprosessointiin, sillä kyseisillä metodeilla voidaan ohittaa osa Apexille asetetuista suoritusrajoitteista.

Jatkokehityksen kannalta hyödylliset kehityskehykset ovat olleet yksi tutkimuskohde. Gupta (2022, s.35) tutki mahdollisuutta hyödyntää ICEM-kehystä (Imperative Code Execution Machine) Salesforce sovellusten kehityksessä. Tuloksien mukaan tämä mahdollistaisi Apex-koodin uudelleenkäytettävyyttä ja vähentäisi kehitystyöhön käytettyä työmäärää.

Salesforcen ohjelmointikieli ja matalan-koodin (low-code) -automaatiotyökalut ovat toinen tuore tutkimuskohde. Ciechan (2023) suoritti tutkimuksen vertaillen tietokantaoperaatioiden

suorituskyvystä hyödyntäen Salesforcen eri matalan-koodin -automaatiotyökaluja, eri UI-kehyksiä ja sykronista Apex-suoritusta. Tuloksissa todettiin, että automaatiotyökaluilla on selkeitä eroja tietokantaoperaatoiden suorituskyvyssä, mutta UI-kehyksien tapauksessa ero ei ole huomattava. Sen sijaan sykroninen Apex-metodi osoittautui suorituskyvyltään toiseksi heikoimmaksi.

2.2 Serialisointi

Kuten aiemmassa luvussa mainittiin, serialisointi on monipuolisesti tutkittu aihealue. Tässä alaluvussa on tarkoituksena käsitellä serialisointia aiempien tutkimuksien pohjalta.

Hericko ym. (2003, s.53) toteuttivat analyysin ja vertailun Javan ja .NET serialisointiprosessien välillä. Tuloksena he totesivat, että binäärimuotoinen serialisointi on tehokkaampaa Javalla, kuin .NET:llä. Sen sijaan XML-muotoinen serialisointi oli tehokkaampaa .NET:llä kuin Javalla.

Toinen merkittävä tutkimus serialisoinnista on Maedan (2012) toteuttama suorituskyvyn vertailu 12 eri ohjelmistokirjasolla XML-, JSON- ja binääriformaattien välillä. Tutkimuksen tuloksena oli vaikeaa arvioida, oliko eri ohjelmistokirjastoilla merkittäviä eroja. Tutkimuksessa binääriformaatti todettiin olevan muistimääräisesti parempi XML- ja JSON-formaatteihin verrattuna.

Serialisointi formaattien soveltuminen mobiilialustoille on ollut myös tutkimuskohteena. Sumaray ja Makki (2012) vertailivat XML- ja JSON-formaattien sekä ProtoBuf ja Apache Thrift binääriformaattien soveltuvuutta mobiilialustalla. Tutkimuksen tuloksena Sumaray ja Makki (2012, s.6) totesivat, että XML-formaattia tulisi välttää mobiilialustoilla ja sen sijaan JSON-formaattia tulisi suosia. Sumaray ja Makki (2012, s.6) totesivat myös, että edellä mainitut binääriformaatit ovat tehokkaampia kuin tekstiformaatit, mutta vaativat mukauttamista mobiilialustalle.

Integroidut ja kolmannen osapuolen serialisointikirjastot ovat myös olleet yksi tutkimuskohde. Nagy ja Kovari (2016, s.429) vertailivat .NET serialisointiprosessien suorituskykyä integroitujen ja kolmannen osapuolen serialisointikirjastojen välillä. Heidän tutkimuksessa

todettiin binääriformaatin olevan paras vaihtoehto suorituskyvyn kannalta. He myös havaitsivat serialisointiprosessin ensimmäisen suorituksen olevan huonoin suorituskyvyltään ja tämän olevan yksi optimoinnin mahdollinen kohde.

Kuten edeltävät tutkimukset osoittavat, serialisointi on ollut ja todennäköisesti tulee olemaan kiinnostava tutkimuskohde. Käytetyt ohjelmointikielet, alustat, mahdolliset ohjelmistokirjastot ja formaatti voivat vaikuttaa koko prosessin suorituskykyyn merkittävästi.

Vaikka serialisointia on tutkittu hyödyntäen useita eri ohjelmointikieliä, ohjelmistokirjastoja ja formaatteja, vastaavaa ei ole vielä toteutettu Salesforce alustalla tai Apex-ohjelmointikielellä. Tämän tutkielman yksi tavoite on toteuttaa alustava katsaus serialisoinnista Apex-ohjelmointikielellä.

2.3 Tutkimusongelma: Serialisointi Apex-ohjelmointikielellä

Apex-ohjelmointikieli on Salesforce alustan tarjoama vahvastityypitetty olio-ohjelmointikieli, joka muistuttaa hyvin paljon Javaa (Zaa ja Verma 2016; Patel ja Chouhan 2016, s.20). Apexia on mahdollista hyödyntää eri toiminnoissa, kuten nappien painalluksissa ja tietuiden päivittämisessä. Apex mahdollistaa myös tietokantaoperaatiot käyttämällä Salesforce Object Query Language (SOQL) kyselykieltä, joka on Structured Query Languege (SQL) kaltainen kyselykieli. (Mathew ja Spraetz 2009). Yleisesti Apex-ohjelmointikieli mahdollistaa kattavien toimintojen toteuttamisen Salesforcessa.

Tutkielman peruslähtökohtana on kaksi tutkijan havaitsemaa tapaa toteuttaa olioiden serialisointi JSON-formaattiin Apex-ohjelmointikielellä. Ensimmäinen havaituista lähestymistavoista perustuu tietueen tietojen asettamiseen avain-arvo-pari rakenteeseen, joka lisätään listaan. Tämä lähestymistapa voidaan käytännössä toteuttaa seuraavalla tavalla:

```
...
List<Object> objects;
List<Map<String,Object>> wrapper =
new List<Map<String,Object>>();
for ( Object tietue : objects ){
```

```
    Map<String,Object> wrap = new Map<String,Object>();
    wrap.put('avain1', tietue.arvo1);
    wrap.put('avain2', tietue.arvo2);
    wrap.put('avain3', tietue.arvo3);
    wrapper.add(wrap);
}
String serialisedJsonString = JSON.serialize(wrapper);
...
```

Toinen olion serialisoinnin toteutusprosessi sisältää kääreluokan hyödyntämisen. Kyseinen luokka on yksinkertainen objekti, joka luodaan Apex-suorituksen ajaksi (Chaudhary 2023). Kääreluokka voidaan toteuttaa seuraavalla tavalla:

```
...
public class JSONWrapper {
    public String avain1;
    public String avain2;
    public Integer avain3;
}
...
```

kääreluokkaa voidaan sen jälkeen käyttää muiden luokkien ja metodien yhteydessä seuraavasti:

```
...
List<Object> objects;
List<JSONWrapper> wrapper = new List<JSONWrapper>();
for ( Object tietue : objects ){
    JSONWrapper wrap = new JSONWrapper();
    wrap.avain1 = tietue.arvo1;
    wrap.avain2 = tietue.arvo2;
    wrap.avain3 = tietue.arvo3;
    wrapper.add(wrap);
```

```
}
String serialisedJsonString = JSON.serialize(wrapper);
...
```

Kummankin prosessin lopputuloksena saavutetaan identtinen JSON-objekti tekstimuodossa:

```
...
{
    "avain1" : "arvo1",
    "avain2" : "arvo2",
    "avain3" : "arvo3"
},
...
```

Laajasti jaettua yhteisymmärrystä ei ole olemassa siitä, kumpaa serialisointimenetelmää tulisi suosia Apex-ohjelmointikielen yhteydessä. Tämän lisäksi tieto serialisointiprosessien mahdollisista skaalautuvuuden ja suorituskyvyen eroista on olematon tällä hetkellä.

3 Tutkimuksen toteuttaminen

Tässä osiossa esitellään käytetty tutkimusmenetelmä ja kuvaillaan tutkielmassa toteutunut tutkimusprosessi.

3.1 Tutkimusmentelmä

Suunnittelutiede on prosessi tietyn ongelman ratkaisemiseen (Hevner ym. 2004, s.82). Tässä tutkielmassa lähtökohtaisena ongelmana ja oletuksena on, että edellisessä kappaleessa esitetyillä serialisointiprosesseilla ei ole eroa skaalautuvuuden ja suorituskyvyn suhteen.

Suunnittelutieteen ensimmäisen ohjeen mukaan, tarkoituksena on toteuttaa artefakti, jonka tulisi käsitellä selkeää ongelmakohdetta (Hevner ym. 2004, s.82). Väitän, että aikaisemmin mainittu tiedonpuute, joka koskee serialisointiprosessien mahdollisia eroja skaalautuvuuden ja suorituskyvyn näkökulmasta, muodostaa selkeän keskeisen ongelmakohdan. Tämä tiedonpuute saattaa johtaa mahdollisesti heikomman serialisointiprosessin valintaan, mikä puolestaan vaikuttaa koko Apex-ohjelmiston suorituskykyyn suoritusajan aikana.

Hevnerin ym. (2004, s.83) mukaan suunnittelutieteessä on tarkoitus toteuttaa teknologia, jolla voidaan vastata tärkeään ja oleelliseen ongelmaan. Kuten ensimmäisessä luvussa mainittiin, Salesforce on rakennettu hyödyntäen multitenant-periaatetta, ja Apex-ohjelmistojen suorituksille on asetettu tiettyjä rajoituksia. Kehittäjien kohtaama haaste liittyy siihen, että heillä ei ole tietoa ainutlaatuisten serialisointiprosessien suorituskykyeroista. Tämän seurauksena ohjelmiston toteuttaminen, joka käyttää serialisointia ilman, että se rikkoo asetettuja rajoituksia Apex-ympäristössä, voi olla ongelmallista.

Hevnerin ym. (2004, s.83) mukaan artefaktin käyttökelpoisuus, laatu ja tehokkuus tulee todistaa tarkoilla evaluointimenetelmillä. Koska tutkielman tarkoituksena on tutkia serialisointiprosessien skaalautuvuuden ja suorituskyvyn eroja, tämän tutkielman pääpaino on artefaktien suoritusajoista luodun aineiston analysoiminen. Evaluointi toteutetaan suorituksien tilastollisia tunnuslukuja verraten. Samakaltaista tilastollista vertailua Ciechan (2023) oli käyttänyt omassa Salesforce aiheisessa tutkielmassaan.

Suunnittelutieteen neljännen periaatteen mukaan, tutkimusmenetelmänän hyödyntäminen tulisi kontributioida uutuuden, yleisyyden ja artefaktin merkityksellisyyden suhteen (Hevner ym. 2004, s.87). Tutkielman tavoitteena on antaa yleistävä katsaus, koska kokonaisvaltaista käsitystä artefaktien skaalautuvuuden ja suorituskyvyn erojen vaikutuksista ja merkityksestä ei vielä ole. Yleistävä katsaus luo pohjan tulevalle erikoistuneemmalle tutkimukselle ja auttaa tutkimusyhteisöä hahmottamaan paremmin artefaktien käyttäytymistä eri skaaloissa ja niiden vaikutuksia suorituskykyyn. Täten kontribuution voidaan olettaa toteutuvan.

Hevner ym. (2004, s.83, 87-88) mukaan niin suunniteltavan artefaktin rakentamisessa kuin evaloinnissa tulisi käyttää järjestelmällisiä menetelmiä. Tutkielmassa kyseistä periaatetta on pyritty seuraamaan artefaktien rakentamisen osalta. Jotta artefaktit antaisivat yleistävät kuvat, luotavissa prosesseissa pyritään käyttämään monipuolisesti Salesforcen tietueilla esiintyviä datatyyppejä. Tämän tarkoituksena on simuloida reaalimaailman prosessia. Tietueilla käytetyt datatyypit ovat:

1. Totuusarvomuuttuja (Boolean)
2. Valuutta (Currency)
3. Päivämäärä (Date)
4. Päivämäärä-Aika (Datetime)
5. Sähköposti (Email)
6. Tunniste (ID)
7. Kokonaisluku (Integer)
8. Valintaluettelo (Picklist)
9. Teksti (String)
10. Aika (Time)

Suunnittelutieteen periatteiden mukaan artefaktin luominen tulisi olla iteratiivinen kehitysprosessi, jotta paras ja tehokkain ratkaisu ongelmaan löydetään (Hevner ym. 2004, s.88). Tässä tutkielmassa iteratiivinen artefaktin kehitysprosessia ei toteuteta, koska tarkoituksena on evaluoida artekfaktien suorituksista saatuja tuloksia, eikä itse luotuja artefakteja.

Suunnittelutieteen seitsämännen periaatteen mukaan, tulokset tulee esittää selkeästi riippumatta kohdeyleisön teknologia orientoituvuudesta (Hevner ym. 2004, s.83). Tämän tutkiel-

man tulokset on oletettu ensisijaisesti hyödyttävän Salesforcen parissa toimivia arkkitehtejä, kehittäjiä ja konsultteja. Siksi tavoitteena on esittää analysoidut tulokset selkeästi ja ymmärrettävästi kyseisille sidosryhmille.

3.2 Tutkimusprosessi

3.2.1 Hiekkalaatikon kuvaus

Tutkielma toteutettiin tutkijan henkilökohtaisessa Salesforce hiekkalaatikossa. Hiekkalaatikkossa ei ollut suoritusajojen aikana muita käyttäjiä tai prosesseja käynnissä, jotka olisivat voineet vaikuttaa suoritusajoihin. Kyseinen hiekkalaatikko on Salesforcen tarjoama ilmainen versio. Hiekkalaatikon versiotiedot ovat esitetty taulukossa 1. Kuten taulukosta 1 käy ilmi, hiekkalaatikossa on tarjolla hyvin rajoitettu muistimäärä. Serialisointiprosesseissa käytetyt tietueet olivat Salesforcen Account eli asiakkuus SObject tyyppiä. Yksi tietue vie noin 2 kilotavua (Salesforce 2023g). Hiekkalaatikon tietokantaan lisättiin 1000 asiakastietuetta, joita oli tarkoitus käyttää serialisointiprosesseissa.

Hiekkalaatiikkoon oli asennettu avoimenlähdekoodin ilmainen kehyssovellus, Nebula logger. Kyseinen kehyssovellus mahdollistaa yksityiskohtaisen lokituksen Apex-luokissa ja automaatityökaluissa, jossa lokit tallennetaan tietokantaan mukautetun sObjektin tietueena (Can 2022). Nebula loggerilla lokitettiin Apex-suoritusajoista kokonaissuoritusaika, käytetyn kekomuistin määrä ja suoritinaika eli CPU-aika. Koska lokitiedot tallennetaan omiin tietueisiin, lokituksen suorittaminen asetti muistimäärän suhteen rajoitteita. Tämän takia Apex-suoritusajon jälkeen kaikki lokitietueet ladattiin xlsx-tiedostoon, jonka jälkeen hiekkalaatikon tietokannasta poistettiin lokitietueet. Tällä varmistettiin, että muistia on vapaana seuraavaa suoritusajoa varten.

Ympäristötieto	Arvo
Organisaation versio	Kehittäjä Version
Instanssi	EU44
Lokaatio	Frankfurt, DE / Paris, FR
Järjestelmä versio	Winter ’24 Patch 11.5
Maksimaalinen muisti	5 mt

Taulukko 1. Salesforce hiekkalaatikon tiedot.

3.2.2 Toteutetut artefaktit

Kuten luvussa 2.3 todettiin, yleisesti serialisointiprosessin voi toteuttaa kahdella tavalla. Joko käyttäen kääreluokkaa tai listaa avain-arvo-pareista. Tutkielmaa varten toteutettiin Apex-luokat eli artefaktit edellä mainituista serialisointiprosesseista. Koska tarkoituksena on luoda yleiskatsaus serialisointiprosessien skaalautuvuuden ja suorituskyvyn eroista, itse artefaktien sisällölliseen arviointiin ei tässä tutkielmassa paneuduta. Sen sijaan luvun 3.1 mukaan, serialisoinnissa käytettyjen tietuiden kenttien datatyypit ovat monipuoliset, jotta serialisointiprosessit voisivat simuloida todellisenmaailman tapausta.

Kääreluokalla suoritettava serialisointiprosessi artefakti oli seuraava:

```
public class WrapperSerializer {
public static String serializeRecords(List<Account> accounts){
    List<JSONWrapper> wrapper = new List<JSONWrapper>();
    for ( Account account : accounts ){
    JSONWrapper wrap = new JSONWrapper();
    wrap.accountId=account.Id;
    wrap.accountNumber=account.AccountNumber;
    wrap.accountName=account.Name;
    wrap.description=account.Description;
    wrap.industry=account.Industry;
    wrap.rating=account.Rating;
    wrap.website=account.Website;
```

```
        wrap.numberOfEmployees=account.NumberOfEmployees;
        wrap.revenue=account.Revenue__c;
        wrap.checked=account.Checked__c;
        wrap.email=account.Email__c;
        wrap.accountTime=account.Time__c;
        wrap.accountDate=account.Date__c;
        wrap.accountDatetime=account.CreatedDate;
        wrapper.add(wrap);
        }
        return JSON.serialize(wrapper);
    }

    public class JSONWrapper {
        public String accountId;
        public String accountNumber;
        public String accountSite;
        public String accountSource;
        public String description;
        public String accountName;
        public Integer numberOfEmployees;
        public String industry;
        public String phone;
        public String rating;
        public String accountType;
        public String website;
        public Boolean checked;
        public Decimal revenue;
        public String email;
        public Time accountTime;
        public Date accountDate;
        public DateTime accountDatetime;
```

```
    }
}
```

Lista avain-arvo-pari luokalla suoritettava serialisointiprosessi artefakti oli seuraava:

```
public class ListMapSerializer {
public static String serializeRecords(List<Account> accounts){
 List<Map<String,Object>> wrapper =
 new List<Map<String,Object>>();
 for ( Account account : accounts ){
    Map<String,Object> wrap = new Map<String,Object>();
    wrap.put('accountId', account.Id);
    wrap.put('accountNumber', account.AccountNumber);
    wrap.put('accountSite', account.Site);
    wrap.put('description', account.Description);
    wrap.put('accountName', account.Name);
    wrap.put('numberOfEmployees', account.NumberOfEmployees);
    wrap.put('industry', account.Industry);
    wrap.put('rating', account.Rating);
    wrap.put('website', account.Website);
    wrap.put('revenue', account.Revenue__c);
    wrap.put('checked', account.Checked__c);
    wrap.put('email', account.Email__c);
    wrap.put('accountTime', account.Time__c);
    wrap.put('accountDate', account.Date__c);
    wrap.put('accountDatetime', account.CreatedDate);
    wrapper.add(wrap);
    }
    return JSON.serialize(wrapper);
    }
}
```

3.2.3 Apex-luokat suoritusajoja varten

Serialisointiprosessien suorittaminen asynkronisesti ja synkronisesti vaati kuuden eri Apex-luokan toteuttamisen. Apex-ohjelmistoa voi kutsua trigger-luokan avulla synkronisesti, asynkronisesti tai SOAP ja REST rajapintojen kautta (Salesforce 2023e). Tässä tutkielmassa Apex-luokat on suuniteltu toimivan samoilla periaatteilla. Suorituksen ajoa aloittavalle Apex-luokalle annetaan haettavien tietueiden määrä kokonaislukuarvona, jota hyödynnetään SOQL-tietokantakutsussa, jossa haetaan serialisoinnissa käytettävät tietueet. Tämän kokonaisluvun tarkoituksena oli toimia skaalaavana arvona. Tämän jälkeen tietueet annetaan Apex-artefakteihin serialisointia varten. Viimeisenä suoritetaan kokonaissuoritusajan, käytetyn CPU-ajan ja kekomuistinmäärän lokitus. Lokitiedot tallennettiin Salesforceen tietueina. Jotta lokitietojen analysoiminen voitaisiin toteuttaa mahdollisimman vaivattomasti, tallennettiin nämä tiedot yhteen tekstikenttään eroteltuna puolipisteellä.

Syknronista suoritusta varten luotiin trigger-luokka, jossa kutsuttiin sykroninsesti suoritettavaa Apex-luokkaa. Trigger-luokan ideana on, että aina kun tietokantaoperaatio suoritetaan sObjektin tietueessa, trigger-luokka käynnistyy joko ennen tai jälkeen, kun tietue on tallennettu (Salesforce 2023h). Tässä tutkielmassa trigger-luokka on asetettu käynnistymään sen jälkeen, kun asiakastietueeseen (sObject) on tehty muutos ja se on tallennettu.

Toteutettu trigger-luokka:

```
trigger AccountTrigger on Account (after update) {
    SynchronousMethod syncMethod = new SynchronousMethod();
    syncMethod.executeSynchronous({n-tietuemäärä});
}
```

Toteutettu synkronisesi suoritettava Apex-luokka:

```
public class SynchronousMethod {
public void executeSynchronous(Integer recordAmount) {
    Long startTime = System.now().getTime();
    List<Account> accountRecordList = [
    SELECT
```

```
    Id,
    AccountNumber,
    Site,
    Description,
    Name,
    NumberOfEmployees,
    Industry,
    Rating,
    Website,
    Revenue__c,
    Checked__c,
    Email__c,
    Time__c,
    Date__c,
    CreatedDate
    FROM Account
    LIMIT :recordAmount];
    String json = {artefaktiluokan kutsuminen};
    Logger.info((System.now().getTime()-startTime)+';'+
    Limits.getHeapSize()+';'+
    Limits.getCpuTime());
    Logger.saveLog();
}
}
```

Asynkroninen suorittaminen vaatii neljä erillistä Apex-luokkaa. Syy tähän on, että Apex-ohjelmointikieli mahdollistaa tällä hetkellä asynkronisen suorittamisen neljällä eri tavalla. Nämä ovat eräajoin suoritettava Apex (Batch Apex), tulevaisuuden metodit (Future Methods), jonotettava Apex (Queueable Apex) ja ajastettu Apex (Scheduled Apex) (Salesforce 2023b).

Eräajoilla suoritettava Apex on Salesforce Lightning alustalla tapahtuva pitkäkestoinen pro-

sessi suurten tietuemäärien käsittelemiseen (Salesforce 2023c). Tietuemäärän voi jakaa pienempiin käsiteltäviin eriin ja Salesforcen perusasetuksena eräajo Apex-ohjelmisto jakaa tietueet 200 tietueen eriin (Marańda, Poniszewska-Maranda ja Szymczyńska 2022). Tässä tutkielmassa tietuemäärän jakamista eriin ei toteutettu, vaan perusasetus muokattiin olemaan aina n-määrä tietuita eli {250, 500, 750, 1000}. Toteutettu eräajo Apex-luokka on seuraava:

```
public class BatchableMethod implements
Database.Batchable<sObject>, Database.Stateful {
    public Integer recordAmount = 0;
    public Long startTime;
    public BatchableMethod(Integer recordAmount){
        this.recordAmount = recordAmount;
    }

public Database.QueryLocator
start(Database.BatchableContext context){
    this.startTime = System.now().getTime();
    return Database.getQueryLocator(
    [SELECT
    Id,
    AccountNumber,
    Site,
    Description,
    Name,
    NumberOfEmployees,
    Industry,
    Rating,
    Website,
    Revenue__c,
    Checked__c,
    Email__c,
    Time__c,
```

```
    Date__c,
    CreatedDate
    FROM Account
    LIMIT : this.recordAmount]);
}

public void execute(Database.BatchableContext context,
    List<Account> scope) {
    String json = {artefaktiluokan kutsuminen};
    Logger.info((System.now().getTime()-this.startTime)+';'+
    Limits.getHeapSize()+';'+
    Limits.getCpuTime());
    Logger.saveLog();
}

public void finish(Database.BatchableContext context){}
}
```

Salesforcen dokumentaation (Salesforce 2023d) mukaan, tulevaisuus metodit ovat taustalla tapahtuva pitkäkestoinen suoritustapa, jossa suoritus asetetaan jonoon ja suoritetaan, kun Salesforcen laskentaresurssit ovat vapautuneet. Näiden käyttäkohteena ovat esimerkiksi verkkopalvelupyynöt tai muut toiminnot jotka halutaan suorittaa omassa säikeessä. Huomiona on, että tulevaisuus metodit sallivat vain primitiiviset tietotyypit parametreinä (Salesforce 2023d). Toteutettu tulevaisuus metodi Apex-luokka on seuraava:

```
public class FutureMethod {
@future
public static void executeAsyncMethod(Integer recordAmount){
    Long startTime = System.now().getTime();
    List<Account> accountRecordList = [
    SELECT
    Id,
```

```
    AccountNumber,
    Site,
    Description,
    Name,
    NumberOfEmployees,
    Industry,
    Rating,
    Website,
    Revenue__c,
    Checked__c,
    Email__c,
    Time__c,
    Date__c,
    CreatedDate
    FROM Account
    LIMIT :recordAmount];
    String json = {artefaktiluokan kutsuminen};
    Logger.info((System.now().getTime()-startTime) +';'+
     Limits.getHeapSize()+';'+
     Limits.getCpuTime());
        Logger.saveLog();
    }
}
```

Tulevaisuusmetodien tapaan, myös jonotettava Apex on taustalla tapahtuva pitkäkestoinen suoritus tapa, mutta jonotettava Apex sallii myös ei-primitiivset tietotyypit paratmetreinä. Tämä sallii myös jonotettavien Apex suoritusten ketjuttamisen (Salesforce 2023f).Toteutettu jonotettava Apex-luokka on seuraava:

```
public class QueueableMethod implements Queueable {
Integer recordAmount;
public QueueableMethod(Integer recordAmount){
```

```
this.recordAmount = recordAmount;
}

public void execute(QueueableContext context) {
Long startTime = System.now().getTime();
List<Account> accountRecordList = [
SELECT
Id,
AccountNumber,
Site,
Description,
Name,
NumberOfEmployees,
Industry,
Rating,
Website,
Revenue__c,
Checked__c,
Email__c,
Time__c,
Date__c,
CreatedDate
FROM Account
LIMIT :this.recordAmount];
String json = {artefaktiluokan kutsuminen};
Logger.info((System.now().getTime()-startTime)+';'+
Limits.getHeapSize()+';'+
Limits.getCpuTime());
Logger.saveLog();
}
}
```

Ajastettu Apex nimensä mukaan mahdollistaa Apex-ohjelmiston suorittamisen tiettynä ajankohtana hyödyntäen Schedulable-rajapintaa (Salesforce 2023a). Ajastuksen asettaminen edellyttää kolmea parametria: ajastuksen yksilöivä nimi, cron-määritelmä ja Schedulable-rajapintaa hyödyntävä Apex-luokka. Toteutettu ajastettu Apex-luokka on seuraava:

```
public class SchedulableMethod implements Schedulable {
Integer recordAmount;
public SchedulableMethod(Integer recordAmount){
    this.recordAmount = recordAmount;
}

public void execute(SchedulableContext context) {
Long startTime = System.now().getTime();
List<Account> accountRecordList = [
SELECT
Id,
AccountNumber,
Site,
Description,
Name,
NumberOfEmployees,
Industry,
Rating,
Website,
Revenue__c,
Checked__c,
Email__c,
Time__c,
Date__c,
CreatedDate
FROM Account
LIMIT :this.recordAmount];
```

```
String json = {artefaktiluokan kutsuminen};
Logger.info((System.now().getTime()-startTime)+';'+
Limits.getHeapSize()+';'+
Limits.getCpuTime());
Logger.saveLog();
}
}
```

3.2.4 Suoritusprosessi

Ciechan (2023, s.158) suoritti 30 suoritusajoa Salesforcessa valitsemillaan kehyksillä. Tässä tutkielmassa päädyttiin suorittamaan 100 suoritusajoa Apex-suorituskontekstia kohden, jotta luotu data olisi mahdollisimaan vertailukelpoista. Yksinkertaisuudessaan, aineiston luominen ja kerääminen toteutettiin seuraavasti. Ensin valittiin suoritettava Apex-luokka {Trigger, Batch, Future, Queueable, Scheduled}. Tämän jälkeen valittiin artefakti, jota kutsutaan suoritettavassa Apex-luokassa. Seuraavaksi Apex-luokkaan asetettiin n-tietumäärä parametri {250, 500, 750, 1000}. Tämän jälkeen suoritettiin 100 erillistä suoritusajoa hiekkalaatikossa. Suoritusajojen päätteeksi tallennetut lokitietueet ladattiin xlsx-tiedostoon ja hiekkalaatikon tietokannasta poistettiin lokitietueet. Suoritusprosessi jatkui iteratiivisesti kunnes n-tietumäärät {250, 500, 750, 1000} olivat suoritettu 100 kertaa molemmilla serialisointi artefekteilla kaikissa Apex-luokissa. Yhteensä suoritusajoja toteutui 4000.

4 Aineiston analysointi

Tässä luvussa analysoidaan suoritusajoista syntynyt aineisto. Tulokset ja kuviot on luotu R:llä ja käytettyt R-koodiskriptit ovat esitetty liitteissä. Ensin käsitellään asynkronisesti luotu aineisto ja sen jälkeen sykronisesti luotu. Tarkastelun kohteena ovat suoritusajojen kokonaissuoritusaika, CPU-aika ja käytetty kekomuistin määrä.

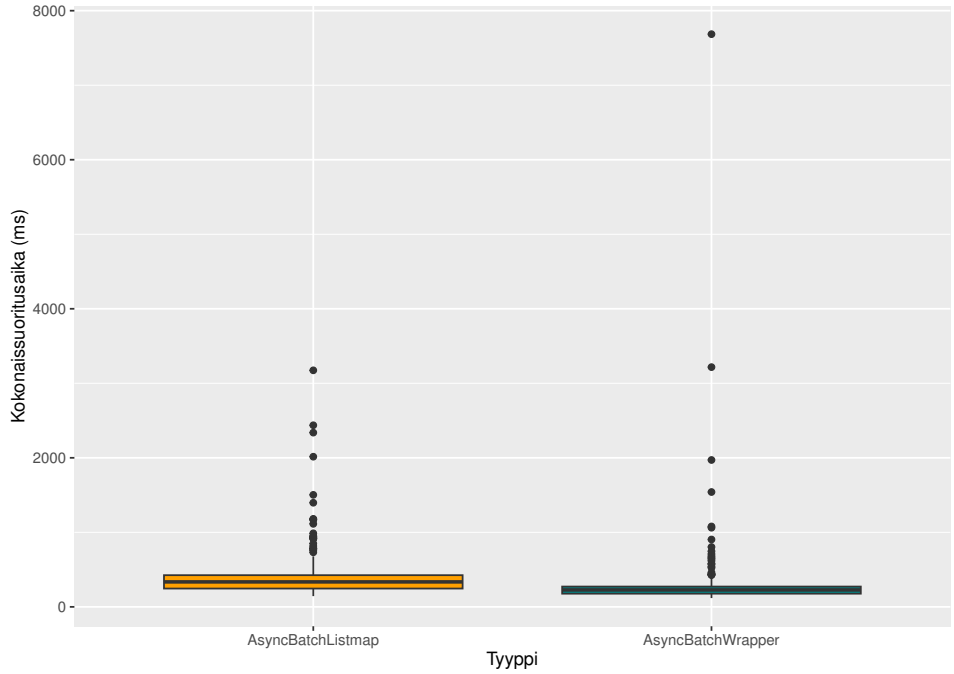
4.1 Asynkroninen aineisto

Asynkronisia suoritusajoja oli yhteensä 3200. 1600 suoritusajoista käytti avain-arvo-lista artekfaktia ja toiset 1600 kääreluokka artefaktia.

4.1.1 Erä Apex (Batch Apex)

Yhteensä erä Apex-luokan suoritusajoja toteutui 800. Luvussa 3.2.2 kuvatulla artefakteilla suoritusajoja suoritettiin 400 artefaktia kohti.

Kuten kuviosta 1 havaitaan, artefaktien välillä ilmenee selkeitä eroja suorituksien kokonaissuoritusaikojen osalta. Molempien artekfatien suorituksissa esiintyy poikkeavia havaintoja. Kääreluokka vaikuttaa ylivoimaisesti tehokkaammalta suoritusaikojen perusteella. Analysoitaessa molempien artefaktien kokonaisaikojen tilastollisia tunnuslukuja taulukossa 2, havaitaan selkeitä eroja. Poikkeavista havainnoista huolimatta kääreluokka -artefakti vaikuttaa olevan tehokkaampi kokonaisaikojen osalta.

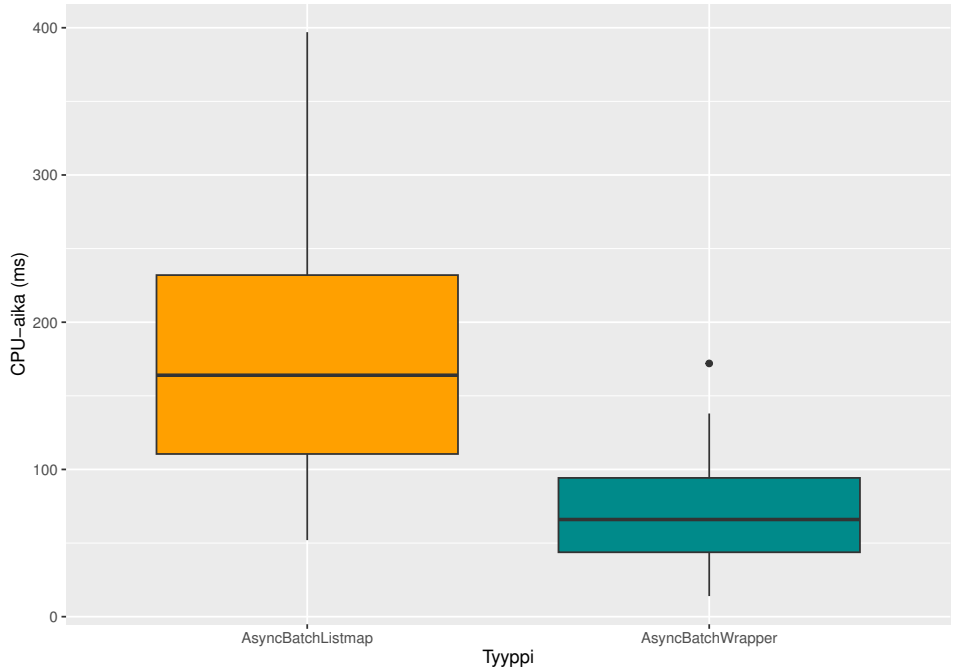


Kuvio 1. Asynkronisten Erä Apex-luokan suorituksien kokonaissuoritusaika.

Kokonaissuoritusaika (ms)		
	Avain-arvo-lista artefakti	kääreluokka artefakti
Min.	144.0	118.0
Q1	245.8	177.0
Mediaani	335.0	227.0
Keskiarvo	386.1	276.1
Q3	425.2	272.5
Max.	3174.0	7685.0

Taulukko 2. Erä Apex-luokkien suorituksien kokonaissuoritusaika tunnusluvut.

Kuviossa 2 esitetään suoritusten toteutuneet CPU-ajat millisekunteina laatikkokaaviossa. Tarkasteltaessa havaitaan, että kääreluokka -artefakti vaikuttaa olevan huomattavasti tehokkaampi verrattuna avain-arvo-lista -artefaktiin, kun tarkastellaan CPU-resursseja. Tämän havainnon tueksi esitetyt tilastolliset tunnusluvut taulukossa 3 vahvistavat kääreluokan ylivoimaisuuden CPU-ajan suhteen.



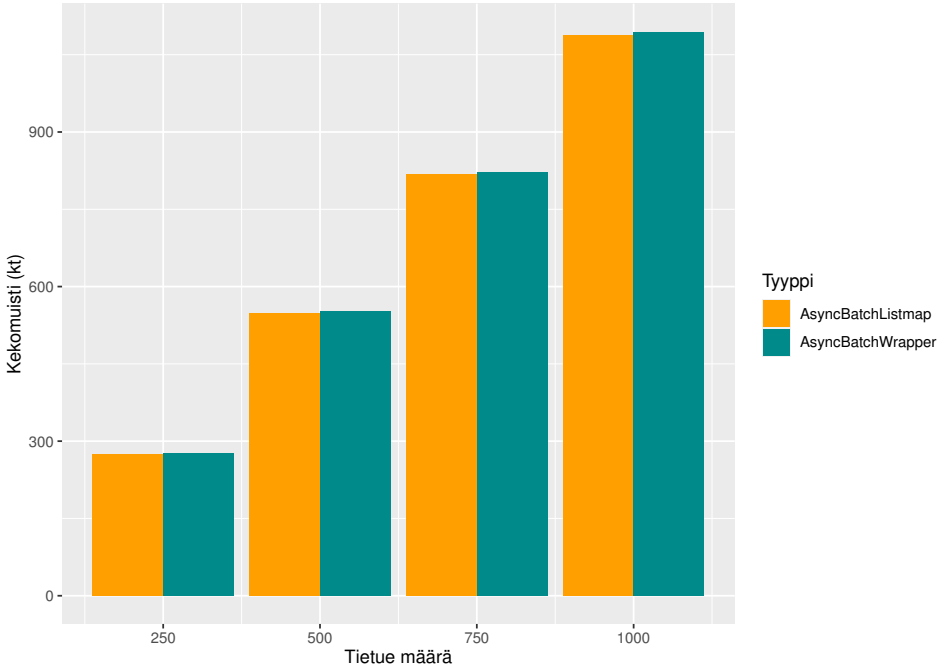
Kuvio 2. Asynkronisten Erä Apex-luokka suorituksien CPU-aika.

CPU-aika (ms)		
	Avain-arvo-lista artefakti	kääreluokka artefakti
Min.	52.0	14.0
Q1	110.5	43.75
Mediaani	164.0	66.0
Keskiarvo	167.7	70.13
Q3	232.0	94.25
Max.	397.0	172.0

Taulukko 3. Erä Apex-luokkien suorituksien CPU-aika tunnusluvut.

Kolmas vertailun kohde on käytetty kekomuisti. Kuviossa 3 voidaan havaita tietueiden määrän vaikuttavan käytetyn kekomuistin määrään lineaarisesti. Taulukkoa 4 tarkasteltaessa huomataan, että kääreluokka artefaktin käyttäneen enemmän kekomuistia kuin avain-arvo-lista artefakti. Kuitenkin käytetyn kekomuistin määrän ero artefaktien välillä on varsin pieni. On mahdollista ettei kyseisellä erolla ole käytännössä olennainen varsinkin, koska havaittu ero

on todella pientä.



Kuvio 3. Asynkronisten Erä Apex-luokan suoritusten kekomuistin määrä (kt) vs. tietue määrä.

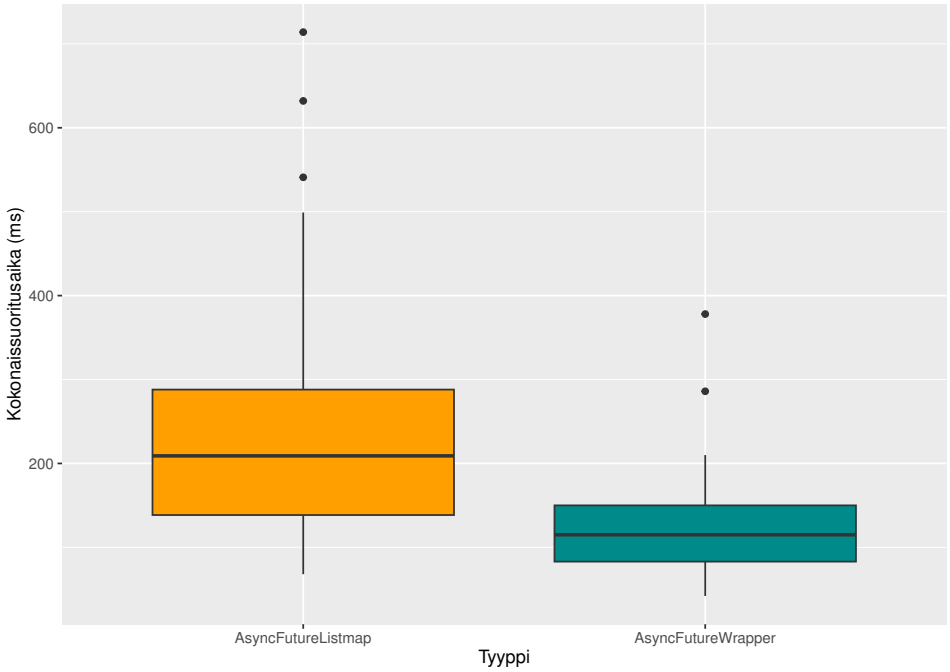
Kekomuisti (kt)		
	Avain-arvo-lista artefakti	kääreluokka artefakti
Min.	274.5	276.6
Q1	479.5	482.6
Mediaani	682.6	686.8
Keskiarvo	681.9	686.2
Q3	885.0	890.4
Max.	1087.9	1094.4

Taulukko 4. Erä Apex-luokkien suorituksien kekomuistin tunnusluvut.

4.1.2 Tulevaisuusmetodit (Future Method)

Tässä alaluvussa on tarkoituksena käsitellä asynkronisen tulevaisuusmetodi Apex-luokan tuloksia. Yhteensä tulevaisuusmetodi Apex-luokan suoritusajoja on 800. Luvussa 3.2.2 kuvatulla artefakteilla suoritusajoja suoritettiin 400 artefaktia kohti.

Kuvio 4 esittää tulevaisuusmetodi Apex-luokan artefaktien kokonaissuoritusaikaa millisekunteina. Samoin kuin erä Apex-luokan tapauksessa, tulosten perusteella voidaan havaita, että kääreluokka -artefakti vaikuttaisi olevan tehokkaampi kokonaissuoritusaika perspektiivissä verrattuna avain-arvo-lista -artefaktiin. Tämä havainto tukee edelleen itseään taulukossa 5, jossa esitetään artefaktien kokonaissuoritusaikaa koskevat tilastolliset tunnusluvut. Näistä voidaan havaita kääreluokka -artefaktin käyttäneen selkeästi vähemmän aikaa suorituksien läpiviennissä.

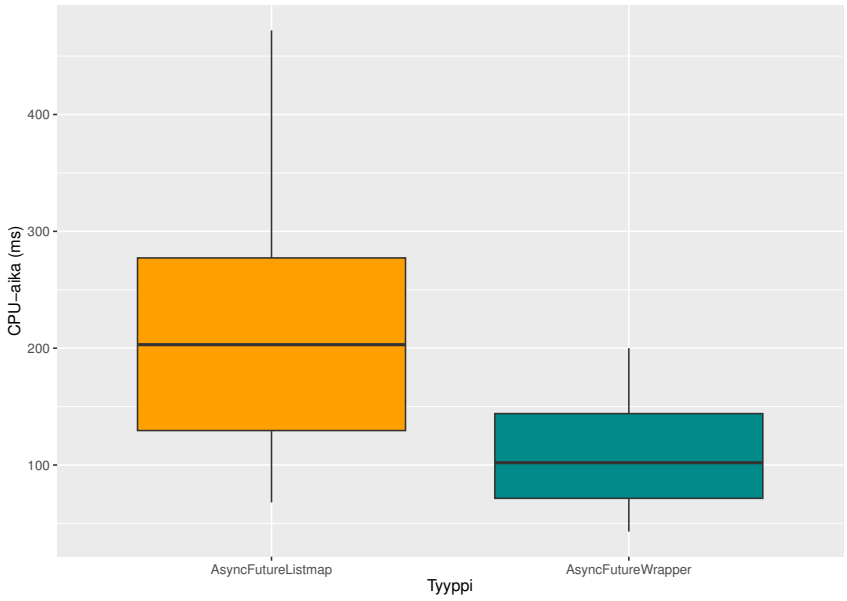


Kuvio 4. Asynkronisten tulevaisuusmetodi Apex-luokkien suorituksien kokonaissuoritusaika.

Kokonaissuoritusaika (ms)		
	Avain-arvo-lista artefakti	kääreluokka artefakti
Min.	68.0	42.0
Q1	138.5	83.0
Mediaani	209.0	115.0
Keskiarvo	213.9	115.2
Q3	288.0	150.0
Max.	714.0	378.0

Taulukko 5. Tulevaisuusmetodi Apex-luokkien suorituksien kokonaissuoritusaika tunnusluvut.

Kuvion 5 perusteella voidaan havaita yhteys, joka on konsistentti aiempien havaintojen kanssa, kuten esitettiin erä Apex-luokan kuviossa 2. Tulokset viittaavat siihen, että kääreluokka -artefakti osoittaa huomattavaa tehokkuutta CPU-aikaresurssien käytössä verrattuna avain-arvo-lista -artefaktiin. Tämä päätelmä on vahvistettavissa myös taulukon 6 esittelemillä tilastollisilla tunnusluvuilla.

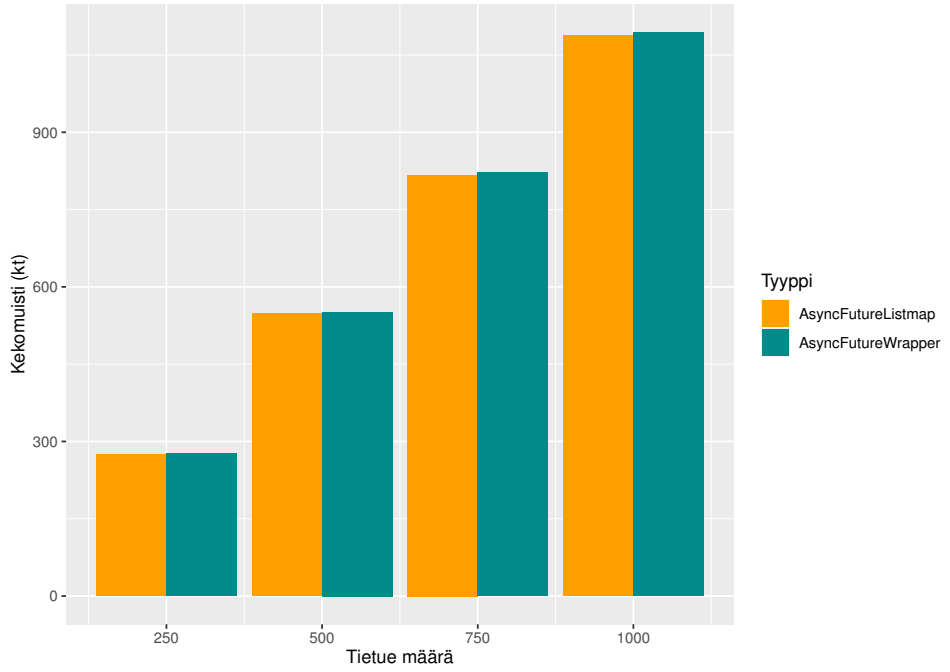


Kuvio 5. Asynkronisten Future Apex-luokkien suorituksien CPU-aika.

CPU-aika (ms)		
	Avain-arvo-lista artefakti	kääreluokka artefakti
Min.	68.0	43.0
Q1	129.5	71.5
Mediaani	203.0	102.0
Keskiarvo	204.2	108.4
Q3	277.2	144.0
Max.	472.0	200.0

Taulukko 6. Future Apex-luokkien suorituksien CPU-aika tunnusluvut

Kuten edellisen luvun kekomuistin esimerkissä, myös tulevaisuusmetodi Apex-luokan artefakti noudattaa samankaltaista käyttäytymistä. Kuvion 6 perusteella voidaan päätellä, että käytetty kekomuisti on erittäin samankaltaista näiden artefaktien välillä. Taulukon 7 tulokset ovat samankaltaiset taulukon 4 tulosten kanssa. Kääreluokka artefakti käyttää enemmän kekomuistia, mutta tämä määrä on huomattavan vähäinen verrattuna avain-arvo-lista -luokka artefaktiin. Kuten Batch Apexin kekomuistin kohdalla on mahdollista, että kyseinen ero ei ole käytännössä olennainen myöskään Future Apexin kohdalla.



Kuvio 6. Asynkronisten Future Apex-luokkien kekomuistin määrä (kt) vs. tietue määrä.

Kekomuisti (kt)		
	Avain-arvo-lista artefakti	kääreluokka artefakti
Min.	274.4	276.5
Q1	479.4	482.5
Mediaani	682.5	686.7
Keskiarvo	681.8	686.0
Q3	884.9	890.2
Max.	1087.8	1094.3

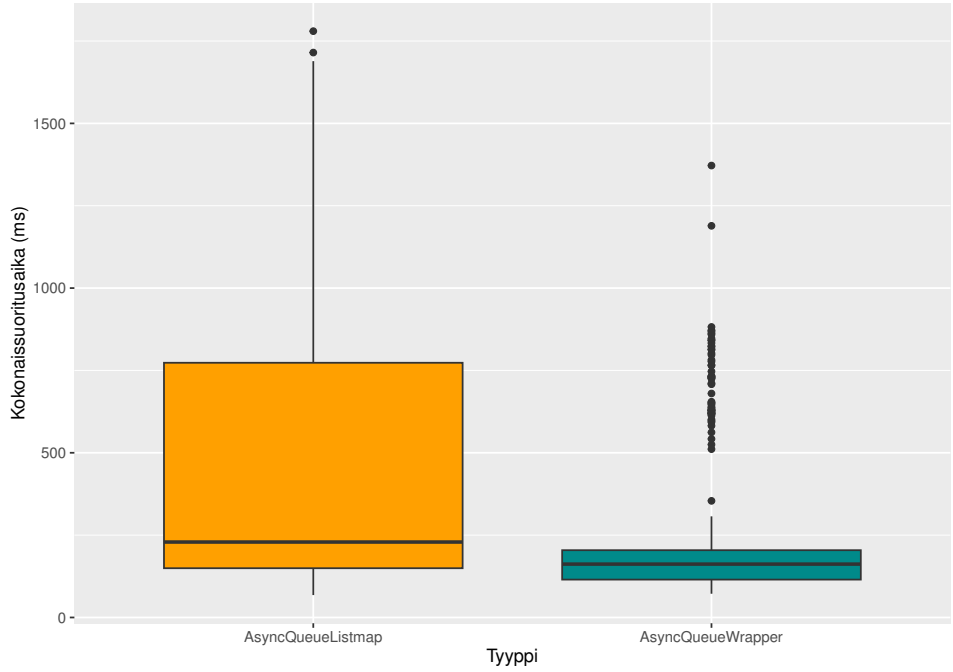
Taulukko 7. Future Apex-luokkien suorituksien kekomuistin tunnusluvut

4.1.3 Jonotettava Apex (Queueable Apex)

Tässä alaluvussa on tarkoituksena tarkastalla jonotettavan Apex-luokan suoritusajojen tuloksia. Yhteensä jonotettavan Apex-luokan suoritusajoja oli 800. Luvussa 3.2.2 kuvatuilla artefakteilla suoritusajoja suoritettiin 400 artefaktia kohti.

Kuvion 7 perusteella voidaan erottaa selkeitä eroavuuksia kääreluokka -artefaktin ja avain-

arvo-lista -artefaktin välillä jonotettavan Apex-luokan tapauksessa. Kuitenkin on syytä havaita, että kääreluokkaan liittyy paljon poikkeavia havaintoja. Tarkasteltaessa taulukkoa 8 voidaan huomata, että näiden luokkien välillä on havaittavissa selkeitä eroja kokonaissuoritusajan tilastollisten tunnuslukujen osalta.

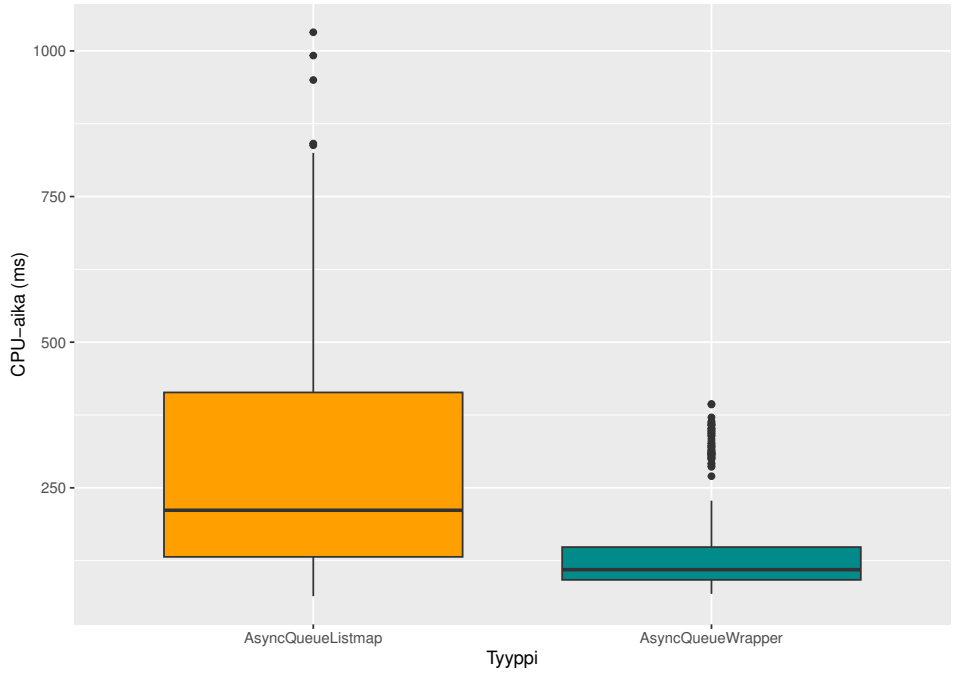


Kuvio 7. Asynkronisen jonotettavan Apex-luokan suorituksien kokonaissuoritusaika.

Kokonaissuoritusaika (ms)		
	Avain-arvo-lista artefakti	kääreluokka artefakti
Min.	68.0	72.0
Q1	149.5	115.0
Mediaani	229.0	115.0
Keskiarvo	402.0	230.2
Q3	773.2	204.2
Max.	1780.0	1372.0

Taulukko 8. Jonotettavan Apex-luokan suorituksien kokonaissuoritusaika tunnusluvut.

Kuten aiemmissa osioissa esitettiin, myös jonotettaviin Apex-luokkiin liittyvissä suorituksissa havaitaan selkeä ilmiö CPU-aikojen osalta. Kuvion 8 ja taulukon 9 perusteella voidaan havaita, että kääreluokkaan liittyvät mittaukset osoittavat alhaisempia arvoja. Tästä syystä voidaan perustellusti väittää, että kääreluokka -artefakti edustaa tehokkaampaa vaihtoehtoa CPU-aikojen suhteen.

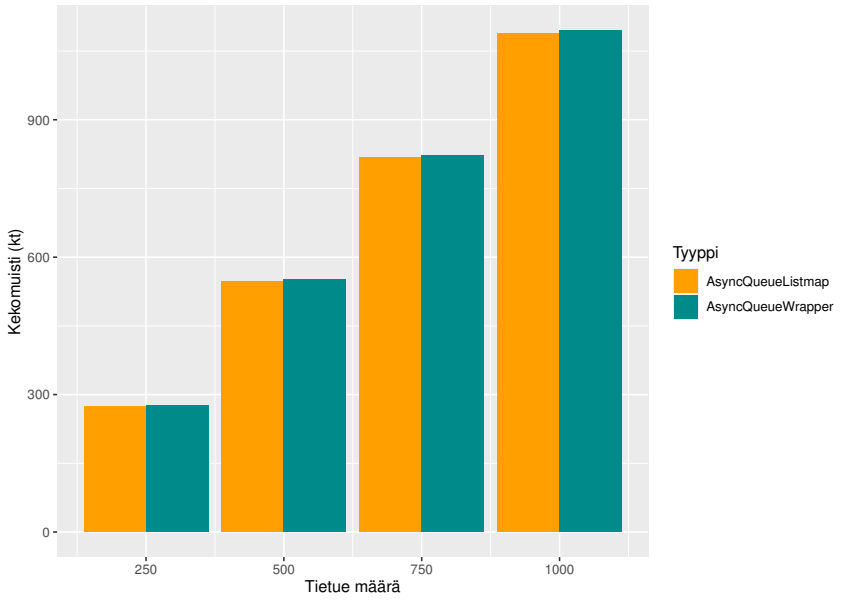


Kuvio 8. Asynkronisen jonotettavan Apex-luokan suorituksien CPU-aika.

CPU-aika (ms)		
	Avain-arvo-lista artefakti	kääreluokka artefakti
Min.	64.0	68.0
Q1	131.5	92.0
Mediaani	211.5	109.5
Keskiarvo	306.6	139.9
Q3	413.8	148.2
Max.	1032.0	394.0

Taulukko 9. Jonotettavan Apex-luokan suorituksien CPU-aika tunnusluvut.

Edellisissä alaluvuissa 4.1.1 ja 4.1.2 huomattiin, että kekomuistin käyttäytyminen oli samankaltaista. Kuvion 9 tarkastelun perusteella voidaan todeta, että kekomuisti skaalautuu tietueiden määrän suhteen samalla tavalla kuin kuviossa 3 ja kuviossa 6 on havaittu. Kun tarkastellaan taulukkoa 10 jonotettavan Apex-luokan kekomuistin tunnuslukujen osalta, huomataan eroavuuksia artefaktien välillä. Kuitenkin, kuten taulukon 4 ja taulukon 7 mukaan, myös taulukon 10 lukujen erot ovat hyvin pieniä. Voidaan siis pitää mahdollisena, ettei tämä ero ole käytännössä olennainen.



Kuvio 9. Asynkronisen jonotettavan Apex-luokan suoritusten kekomuistin määrä (kt) vs. tietue määrä.

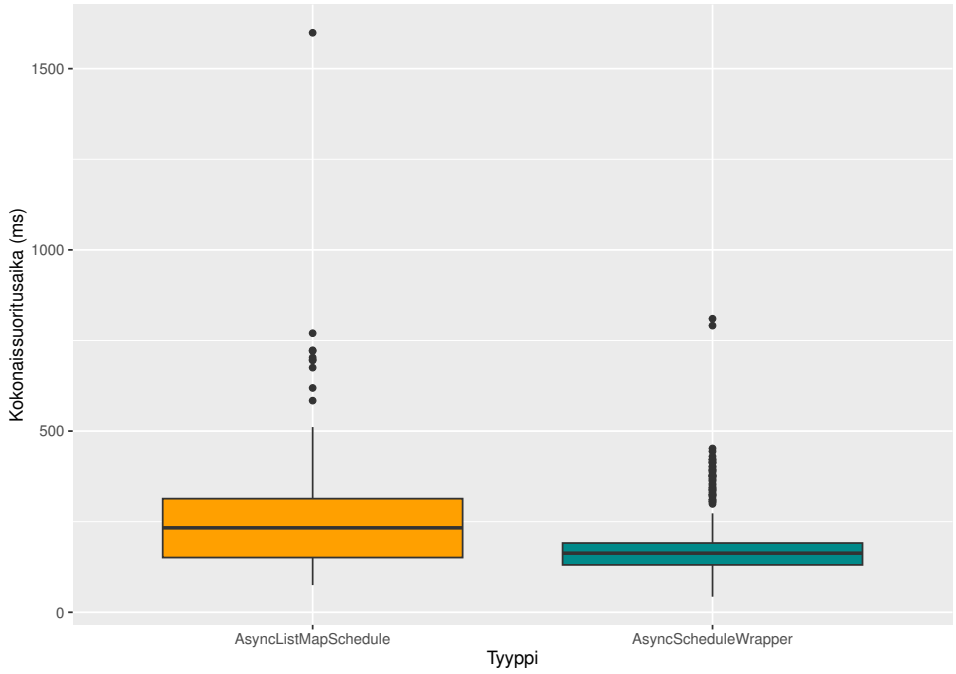
Kekomuisti (kt)		
	Avain-arvo-lista artefakti	kääreluokka artefakti
Min.	274.4	276.5
Q1	479.4	482.5
Mediaani	682.5	686.7
Keskiarvo	681.8	686.1
Q3	884.9	890.3
Max.	1087.8	1094.3

Taulukko 10. Jonotettavan Apex-luokan suorituksien kekomuistin tunnusluvut.

4.1.4 Ajastettu Apex (Scheduled Apex)

Yhteensä ajastettuja Apex-luokan suoritusajoja on 800. Luvussa 3.2.2 kuvatuilla artefakteilla suoritusajoja suoritettiin 400 artefaktia kohti.

Kuviota 10 analysoitaessa havaitaan, että molempien artefaktien kokonaissuoritusajoissa esiintyy poikkeavuuksia. Kuitenkin voidaan päätellä, että kääreluokkaan perustuva artefakti osoitti tehokkaampaa suorituskykyä verrattuna avain-arvolistaa hyödyntävään artefaktiin. Tätä havaintoa vahvistavat myös taulukossa 11 esitetyt tilastolliset tunnusluvut, jotka osoittavat kääreluokka artefaktin kokonaissuoritusaikojen alhaisemmat arvot.

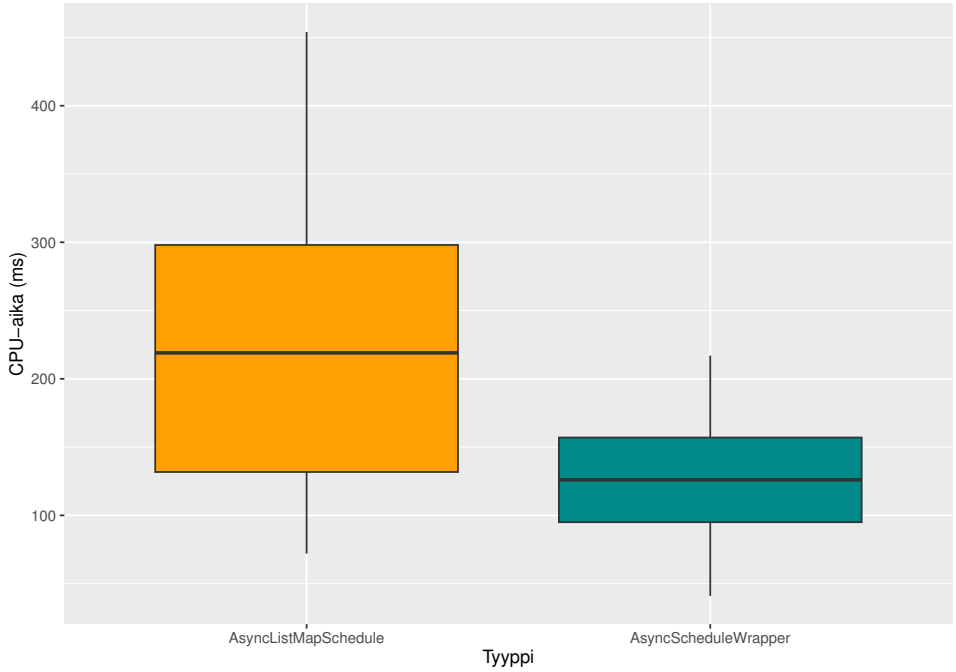


Kuvio 10. Asynkronisen ajastetun Apex-luokan suorituksien kokonaissuoritusaika.

Kokonaissuoritusaika (ms)		
	Avain-arvo-lista artefakti	kääreluokka artefakti
Min.	75.0	43.0
Q1	151.0	130.8
Mediaani	233.0	163.0
Keskiarvo	241.2	175.5
Q3	313.5	191.0
Max.	1599.0	810.0

Taulukko 11. Ajastetun Apex-luokan suorituksien kokonaissuoritusajan tunnusluvut.

Kun tarkastellaan ajastetun Apex-luokan suorituksia CPU-aikaresurssien näkökulmasta, voidaan havaita selkeä ilmiö. Kuten ilmenee kuvion 11 perusteella, kääreluokkaan perustuvan artefaktin käyttämät CPU-resurssit vaihtelevat vähemmän kuin avain-arvo-lista artefaktin. Taulukkoa 12 tarkasteltaessa huomataan, että kääreluokka -artefaktin tilastolliset tunnusluvut ovat matalempia. Tästä voi tehdä päätelmän, että kääreluokka -artefakti tarjoaa tehokkaamman ratkaisun.

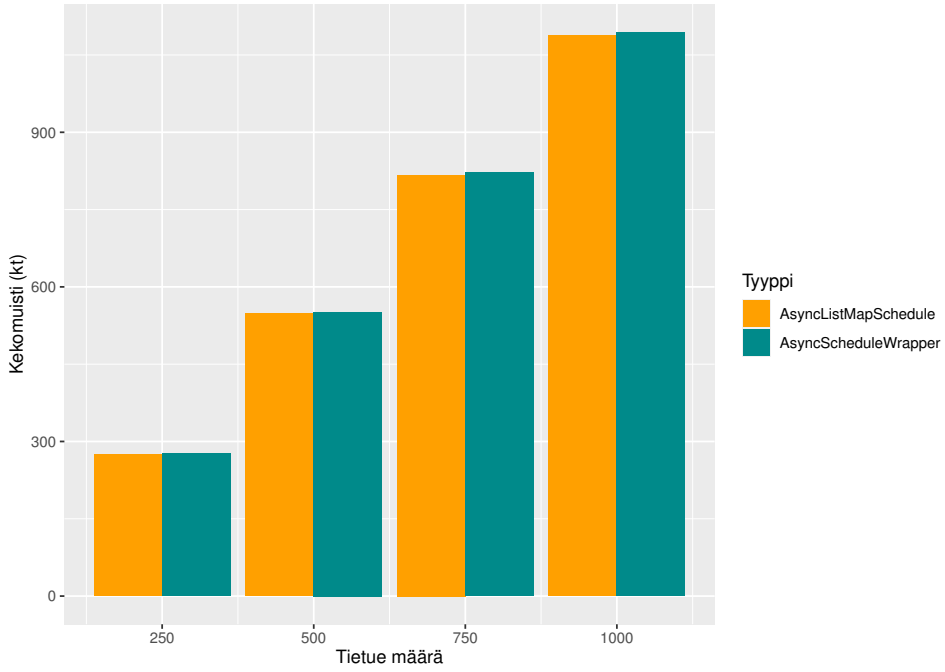


Kuvio 11. Asynkronisen ajastetun Apex-luokan suorituksien CPU-aika.

CPU-aika (ms)		
	Avain-arvo-lista artefakti	kääreluokka artefakti
Min.	72.0	41.0
Q1	131.8	95.0
Mediaani	219.0	126.0
Keskiarvo	217.6	125.6
Q3	298.0	157.0
Max.	454.0	217.0

Taulukko 12. Ajastetun Apex-luokan suorituksien CPU-aika tunnusluvut.

Kuten aikaisemmissa osioissa havaittiin, samanlainen ilmiö havaitaan myös ajastetun Apex-luokan kekomuistinkäytön yhteydessä. Kuviosta 12 ilmenee, että avain-arvo-lista-luokkaan perustuva artefakti suoriutuu hieman pienemmällä kekomuistin käytöllä. Sen sijaan kääreluokkaan perustuva artefakti vaatii hieman enemmän kekomuistia. Taulukko 13, joka sisältää kekomuistin tilastolliset tunnusluvut, tukee tätä havaintoa. Kuitenkin tässäkin tapauksessa voidaan todeta, että artefaktien välillä tapahtuva kekomuistimäärän vaihtelu on varsin pientä, ja on mahdollista, ettei tämä ero ole käytännössä relevantti.



Kuvio 12. Asynkronisten Schedule Apex-luokkien kekomuistin määrä (kt) vs. tietue määrä.

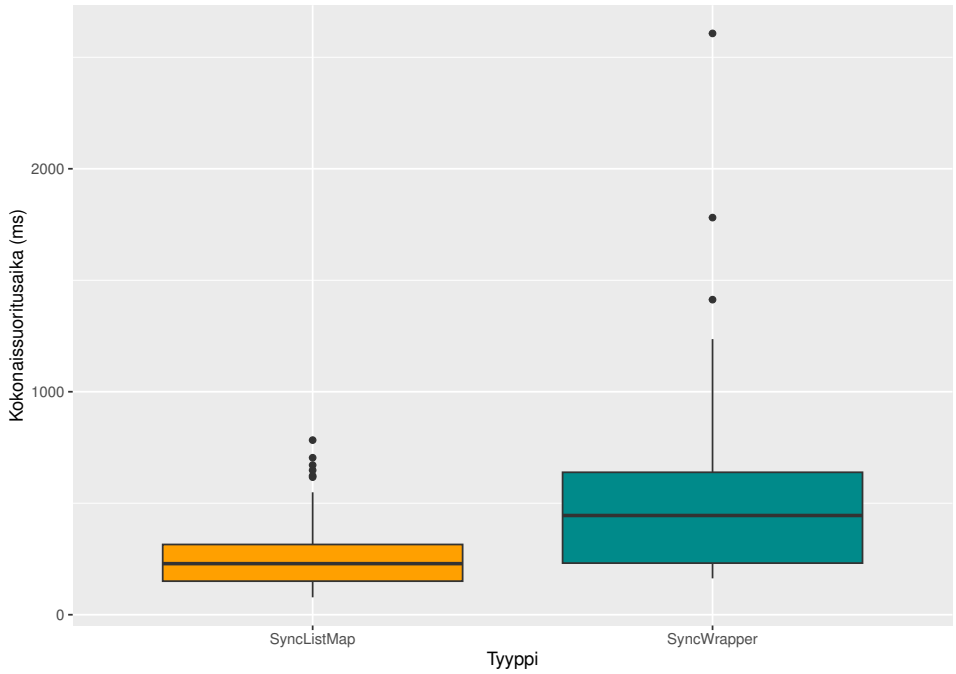
Kekomuisti (kt)		
	Avain-arvo-lista artefakti	kääreluokka artefakti
Min.	274.4	276.5
Q1	479.4	482.5
Mediaani	682.5	686.7
Keskiarvo	681.8	686.1
Q3	884.9	890.3
Max.	1087.8	1094.3

Taulukko 13. Ajastetun Apex-luokan suorituksien kekomuistin tunnusluvut.

4.2 Synkroninen aineisto

Tässä luvussa tarkastellaan synkronisella suoritustavalla luotua aineistoa. Yhteensä suoritusajoja tapahtui 800, joista 400 tapahtui luvussa 3.2.2 kuvatulla avain-arvo-lista luokalla ja toiset 400 kääreluokalla.

Toisin kuin edellisessä luvussa käsitellyissä asynkronisissa suorituksissa, synkronisten suoritusten kokonaissuoritusaika vaikuttaa olevan avain-arvo-lista -pohjaisen artefaktin kanssa tehokkaampi. Kuten kuvio 13 osoittaa, kokonaissuoritusaika näyttää tässä tapauksessa olevan vakaa eikä sisällä suuria vaihteluita. Tätä vahvistaa myös taulukon 14 esittämät tilastolliset tunnusluvut, joista käy ilmi, että avain-arvo-lista -pohjainen artefakti on tehokkaampi vaihtoehto.

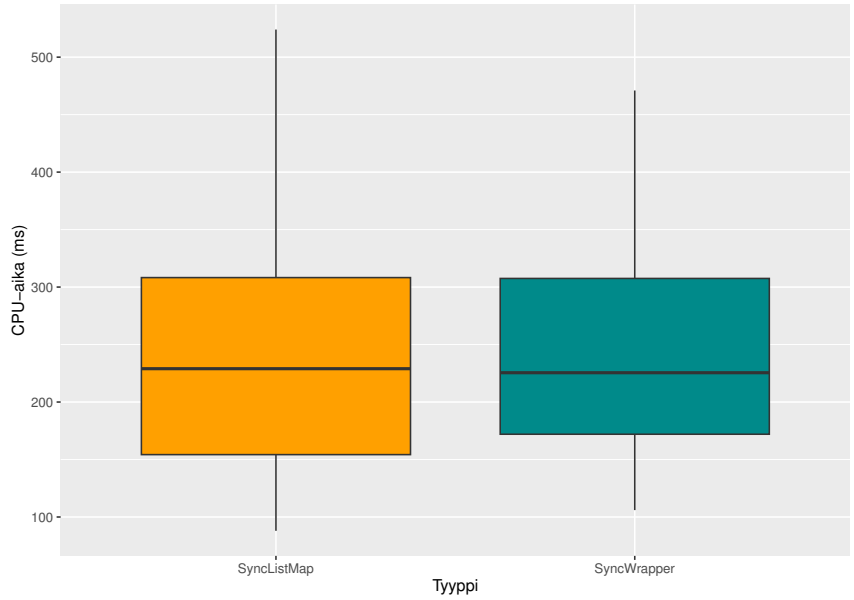


Kuvio 13. Synkronisen Apex-luokan suorituksien kokonaissuoritusaika.

Kokonaissuoritusaika (ms)		
	Avain-arvo-lista- artefakti	kääreluokka artefakti
Min.	78.0	163.0
Q1	150.5	231.5
Mediaani	229.0	445.0
Keskiarvo	233.8	474.8
Q3	315.0	638.8
Max.	783.0	2607.0

Taulukko 14. Synkronisen Apex-luokan suorituksien kokonaissuoritusaika tunnusluvut.

Synkronisten Apex-luokkien suorituksissa havaitaan selkeitä eroja CPU-resurssien käytössä verrattuna edellisen luvun asynkronisten suoritusten CPU-aikoihin. Kuten Kuvio 14 osoittaa, molempien artefaktien osalta CPU-ajan käyttö on ollut samankaltaista. Taulukossa 15 esitetyistä tilastollisista tunnusluvuista voidaan huomata, että kääreluokka -artefakti on suoriutunut keskimäärin paremmin. Kuitenkin kyseinen artefakti on myös minimiarvon ja alakvartiilin mukaan käyttänyt enemmän CPU-aikaresursseja, kuin avain-arvo-lista -artefakti.

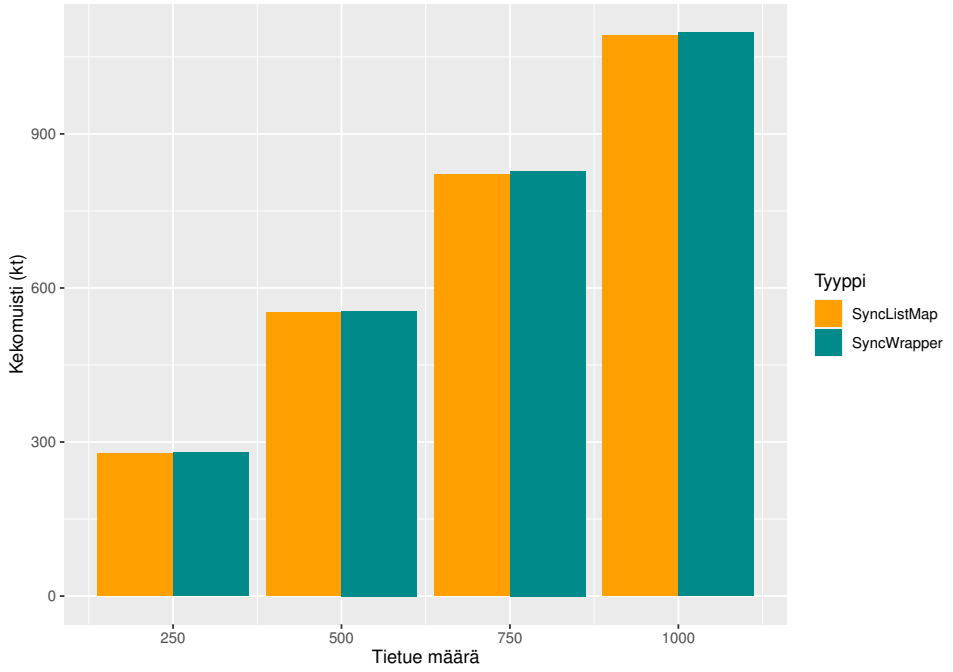


Kuvio 14. Synkronisen Apex-luokan suorituksien CPU-aika.

CPU-aika (ms)		
	Avain-arvo-lista artefakti	kääreluokka artefakti
Min.	88.0	106.0
Q1	154.2	172.0
Mediaani	229.0	225.5
Keskiarvo	230.5	240.8
Q3	308.2	307.5
Max.	524.0	471.0

Taulukko 15. Synkronisen Apex-luokan suorituksien CPU-aika tunnusluvut.

Kuten aikaisemman osion asynkronisten suoritusaikojen tapauksessa, myös synkronisten suoritusten yhteydessä havaitaan samankaltaista ilmiötä. Kuvion 15 perusteella havaitaan, että kääreluokkaan perustuvan artefaktin ja avain-arvo-listaa käyttävän artefaktin välinen ero kekomuistin käytössä on varsin vähäinen. Taulukon 16 esittämät tilastolliset tunnusluvut vahvistavat tätä havaintoa. Kuten aikaisemmassa asynkronisten suoritusaikojen kekomuistin analyysissä, myös synkronisten suoritusten osalta tunnuslukujen vaihtelu on hyvin vähäistä. Voidaan siis pitää mahdollisena, ettei vaihtelu ole käytännössä relevantti.



Kuvio 15. Synkronisen Apex-luokan kekomuistin määrä (kt) vs. tietue määrä.

Taulukko 16. Synkronisen Apex-luokan suorituksien kekomuistin tunnusluvut

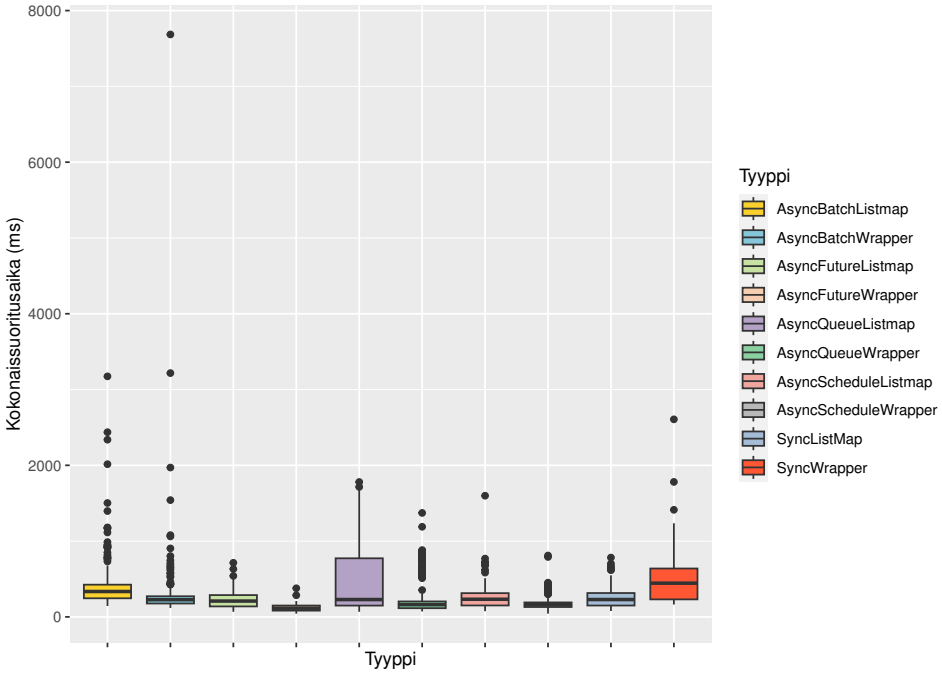
Kekomuisti (kt)		
	Avain-arvo-lista artefakti	kääreluokka artefakti
Min.	278.2	280.3
Q1	483.2	486.3
Mediaani	686.3	690.5
Keskiarvo	685.6	689.9
Q3	888.7	894.1
Max.	1091.7	1098.2

Taulukko 17. Synkronisen Apex-luokan suorituksien kekomuistin tunnusluvut.

4.3 Aineisto-analyysin johtopäätökset

Tässä luvussa pyritään esittämään päätelmät edellisissä kappaleissa tehtyjen havaintojen pohjalta. Tätä varten aineisto on integroitu yhtenäiseksi kokonaisuudeksi, mikä mahdollistaa selkeän vertailun.

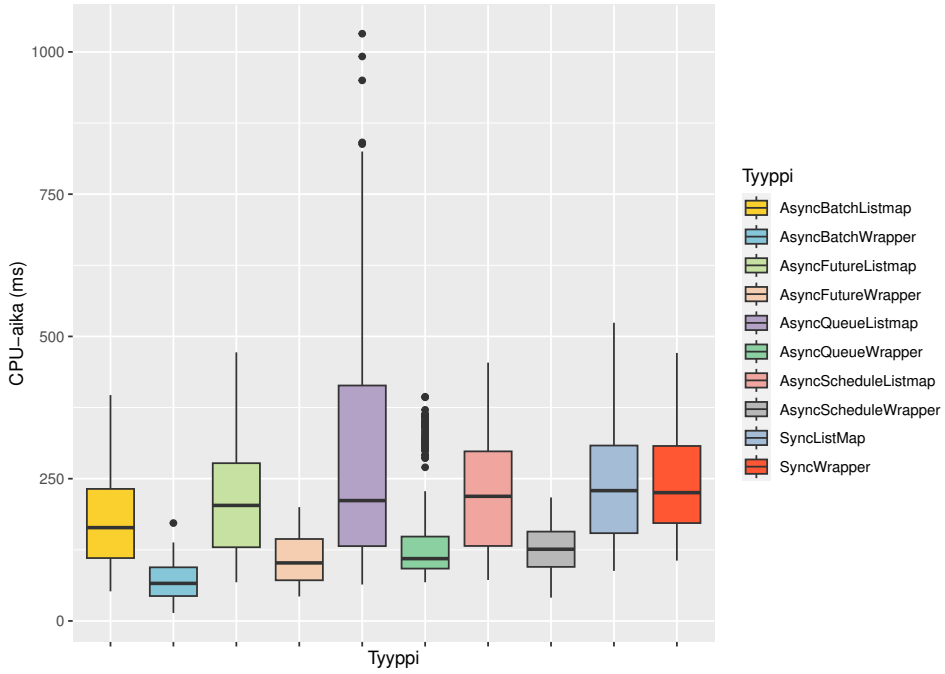
Kuvio 16 esittää kattavasti kaikkien Apex-luokkien kokonaissuoritusaikoja. Kuvio tarjoaa selkeän näkymän useisiin havaittavissa oleviin ilmiöihin. Jokaisen yksittäisen Apex-luokan osalta esiintyi poikkevia havaintoja suoritusaikojen osalta. Näiden poikkeamien todennäköinen syy liittyy aiemmin esitettyyn Salesforcen monikäyttäjäperiaatteeseen (multitenant), joka jakaa laskentaresurssit eri organisaatioiden kesken. Vaikka poikkeavia havaintoja on paljon, kuviossa on kuitenkin toinen selkeä suuntaus. Kääreluokan kokonaissuoritusaika oli systemaattisesti pienempi kaikissa muissa tapauksissa paitsi synkronisessa suorituksessa. Tästä voidaan päätellä, että havainto tarjoaa vahvan tuen sille, että Apexin asynkronisen ja synkronisen suorituskyvyn välillä artefakteilla on selkeitä eroja serialisointiprosessien kokonaissuoritusaikaa tarkasteltaessa.



Kuvio 16. Kaikkien Apex-luokan suorituksien kokonaissuoritusaika.

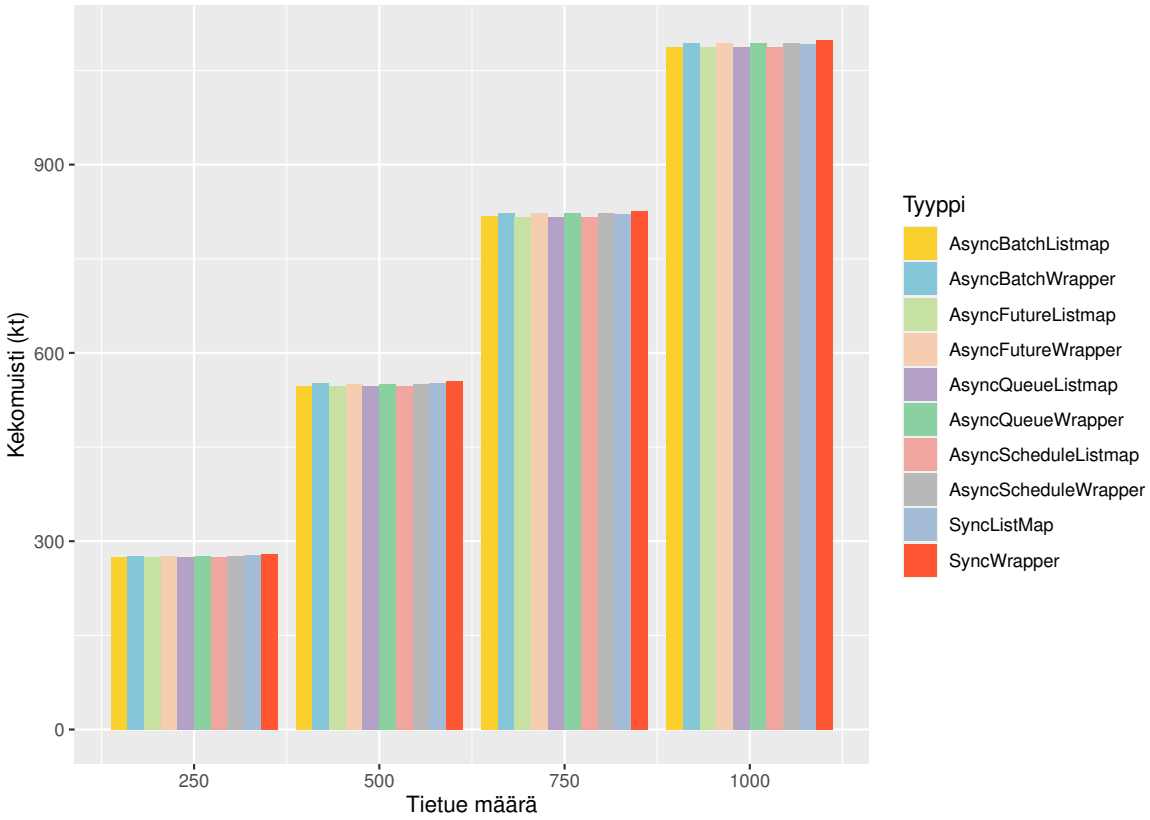
Kuviossa 17 esitetään kaikkien suoritusten CPU-ajat kattavasti. Toisin kuin kokonaissuoritusaikojen suhteen, CPU-aikojen yhteydessä havaitaan poikkeamia ainoastaan eräajo Apex-luokan kääre -artefaktin, jonotettavan avain-arvolista -artefaktin ja jonotettavan kääre -artefaktin yhteydessä. Näistä poikkeamista huolimatta kuvio paljastaa selkeän ilmiön. Kääreluokka artefakti on osoittautunut keskimäärin tehokkaammaksi kaikissa asynkronisissa suorituksissa sekä synkronisessa suorituksessa. Tämän perusteella voidaan tehdä päätelmä, että CPU-

resurssikäytön suhteen kääreluokan artefakti edustaa suorituskyvyltään tehokkaampaa vaihtoehtoa serialisointiprosesseissa.



Kuvio 17. Kaikkien Apex-luokkien suorituksien CPU-ajat.

Kuviossa 18 esitetään kaikkien Apex-luokkien suorituksien kekomuistin määrä suhteessa tietueiden määrään. Kuten kuvion perusteella voidaan huomata, kekomuistin käytössä havaitaan hyvin vähäisiä eroja artefaktien välillä. Myös asynkronisten ja synkronisten suoritusten välillä ilmenee varsin vähäistä eroa. Kuitenkin kääreluokan artefakti näyttää käyttäneen enemmän kekomuistia. Kuvion perusteella on hankala arvioida, onko tämä ero keskeinen ongelma suorituskyvyn kannalta. Sen sijaan voidaan todeta, että kekomuistin skaalautuvuus on samankaltainen eri suorituskonteksteissa molemmilla artefakteilla.



Kuvio 18. Kaikkien Apex-luokan suorituksien kekomuistin määrä (kt) vs. tietue määrä.

Tulosten perusteella tehtävissä johtopäätöksissä huomioidaan aineistoanalyysin tulokset, jotka viittaavat siihen, että kokonaissuoritusajan ja CPU-ajan näkökulmasta suurimassa osassa suorituksista kääreluokan artefakti osoittautuu suorituskyvyltään tehokkaammaksi serialisointiprosessissa verrattuna avain-arvo-lista -artefaktiin. Näin ollen luvussa 3.1 esitetty oletus, jonka mukaan kääreluokan ja avain-arvo-listan artefaktit eivät poikkea toisistaan suorituskyvyn suhteen, on havaittu virheelliseksi. Sen sijaan on mahdollista väittää, että skaalautuvuuden osalta artefaktit skaalautuvat samankaltaisesti. Voidaan pitää myös hyvin mahdollisena, että havaittu eroavuus ei ole käytännössä relevantti. Tämän varmistaminen kuitenkin vaatisi tarkempaa tilastollista analysointia.

5 Pohdinta

Luku 1 toimi tutkielman lähtökohtana, esittäen kaksi keskeistä tutkimuskysymystä. Nämä kysymykset olivat: Miten serialisointiprosessit, jotka hyödyntävät kääreluokkaa ja avain-arvo-pari -listaa, skaalautuvat tietuemäärän kasvaessa? ja toiseksi: Mitkä suorituskykyerot ilmenevät eri asynkronisten ja synkronisten Apex-suoritustapojen välillä, kun ne liittyvät edellä mainittuihin serialisointiprosesseihin? Luvussa 2 esiteltiin katsaus Salesforceen ja serialisointiin sekä näiden aiempiin tutkimuksiin. Lisäksi tarkasteltiin, miten serialisointi voidaan toteuttaa Salesforcen Apex-ohjelmointikielellä. Luvussa 3 avattiin käytetty tutkimusmenetelmä yksityiskohtaisesti ja kuvattiin tutkimusprosessin eteneminen. Luku 4 keskittyi aineiston analysointiin ja johtopäätöksien esittämiseen, joiden pohjalta voitiin vastata asetettuihin tutkimuskysymyksiin.

Alkuperäisenä oletuksena oli, ettei avain-arvo-listan ja kääreluokkan serialisointitavoilla ollut minkäänlaisia eroja skaalautuvuuden ja suorituskyvyn suhteen. Vaikka skaalautuvuuden suhteen havaittiin eroa, vaihtelu ei ollut keskimäärin suurta, eikä käytännössä tämä saata olla relevantti. Selkeä ero ilmeni kokonaissuoritusajan ja CPU-ajan suhteen eri asynkronisten ja synkronisten suoritustapojen välillä artefaktien osalta.

Kokonaissuoritusajan osalta kääreluokka -artefakti, yhdessä tulevaisuusmetodin ja ajastetun Apex-luokan kanssa, osoittautui keskimäärin tehokkaimmiksi asynkronisissa suorituksissa. Sen sijaan synkronisissa suorituksissa avain-arvo-lista -artefakti suoriutui keskimäärin paremmin verrattuna kääreluokka -artefaktiin. CPU-ajan suhteen kääreluokka artefakti yhdessä batch Apex-luokan kanssa saavutti keskimäärin parhaimmat tulokset. Tämän jälkeen keskimäärin hyvällä CPU-ajalla suoriutuivat tulevaisuusmetodi ja ajastettu kääreluokka -artefakti.

Aineistoanalyysin pohjalta voidaan todeta alkuperäinen oletus vääräksi. Näiden havaintojen perusteella voidaan väittää, että tutkimuskysymyksiin on vastattu ja asetetut tavoitteet on saavutettu.

Vaikka tutkielmassa saavutettiin selkeitä tuloksia, on tärkeää tunnistaa tutkielmaan liittyvät selkeät rajoitteet. Yksi keskeinen rajoite liittyi käytettyyn hiekkalaatikkoon. Vaikka Salesforce-alustan eri hiekkalaatikot perustuvat samoihin periaatteisiin, valitun hiekkalaatikon muistira-

joitukset asettivat esteitä suuren datamäärän täysimääräiselle hyödyntämiselle. On mahdollista, että suuremmalla datamäärällä ja lokitettujen suoritusajojen määrällä olisi voitu saavuttaa laajempi ja vertailukelpoisempi aineisto, mikä olisi lisännyt tulosten luotettavuutta. Rajoitteista huolimatta tutkielman vahvuutena voidaan pitää uutta näkökulmaa, joka tuo esiin aiemmin tutkimattoman ulottuvuuden Salesforce-palvelussa.

Tutkielman toteutukseen vaikutti tutkijan henkilökohtainen kokemus Salesforce-alustasta ja Apex-ohjelmointikielestä. Aiempi työkokemus näiden teknologioiden parissa antoi luonnollisen kiinnostuksen tutkia tarkemmin tutkielman aihepiiriä. On tärkeää tunnustaa, että tutkijan kokemus alustasta ja ohjelmointikielestä vaikutti tutkimuksen taustalla oleviin oletuksiin ja lähestymistapoihin. Yksi vaikutus oli, että tulokset esitettiin kuvailevasti ja johtopäätökset esitettiin practitioner-pohjalta, ja ne pohjautuivat suoraan käytännön kokemukseen ja näkemyksiin. Vaikka tutkielmassa ei tehty aineiston analyysia tilastollisen testien kautta, tulokset voivat mahdollisesti tarjota arvokasta tietoa, jotka ovat suoraan sovellettavissa käytännön ohjelmistokehitykseen.

6 Johtopäätökset

Tämä tutkielma tarjosi alustavan tarkastelun serialisointiprosesseista Salesforcen Apex-ohjelmointikielellä. Tämä valinta perustui havaintoon, että vastaavaa tutkimusta tai toteutusta ei ollut aiemmin toteutettu kyseisellä ohjelmointikielellä. Tulosten valossa voitiin osoittaa, että serialisointiprosessien suorituskyvyillä on eroja asynkronisessa ja synkronisessa kontekstissa. Tutkituilla asynkronisilla ja synkronisilla prosesseilla on omat vahvuudet sekä heikkoudet ja vaativat kattavan ymmäryksen prosessien toiminnoista (Marańda, Poniszewska-Maranda ja Szymczyńska 2022). Täten on mahdotonta määrittellä tutkimuksen kontekstissa, onko kannattavaa suosia yhtä suorituskontekstia toisen sijasta. Sen sijaan väitän aineistoanalyysin perusteella kääreluokan olevan tehokkaampi ratkaisu asynkronisessa kontekstissa. Kuitenkin synkronisessa kontekstissa avain-arvo-lista -rakenteen sopeutui kokonaissuorituksen suhteen paremmin. Skaalautuvuuden suhteen hainnot olivat samankaltaisia artefaktien ja suorituskontekstien välillä.

Tutkielman keskeinen tavoite oli muodostaa yleistävä kuvaus serialisointiprosesseista, jotka suoritetaan Apex-ohjelmointikielellä. Saavutetut tulokset ovat erityisen merkitykselliset Apex-ohjelmistokehittäjille, sillä ne tarjoavat selkeän käsityksen suorituskyvyllisistä eroista näiden prosessien välillä. Kyseinen tieto voi olla hyödyllinen päätöksenteossa ja suunnittelussa, mahdollistaen kehittäjille tehokkaamman lähestymistavan serialisointiprosessien toteuttamiseen Apex-ohjelmointikielellä.

Tutkielma heijasteli kokonaisvaltaista pyrkimystä ymmärtää serialisoinnin dynamiikkaa ja sen vaikutusta suorituskykyyn, ja täten se avaa mahdollisuuksia jatkotutkimukselle tällä alueella. Pyrkimys luoda yleistävä kuva antaa mahdollisuuden syventää tutkimusta esimerkiksi tarkastelemalla tiettyjen rajapintojen vaatimia JSON-objekteja. Tässä yhteydessä voitaisiin harkita vastaavien artefaktien systemaattista generointia, jonka avulla voitaisiin systemaattisesti arvioida niiden suorituskykyä. Tämä laajennettu tutkimus voi tuottaa lisäarvoa Apex-ohjelmointikielen JSON-rakenteiden käsittelyyn liittyvistä suorituskykyvyllisistä eroista ja syventää tietämystämme eri artefaktien tehokkuudesta kyseisessä kontekstissa.

Toinen mahdollinen jatkotutkimuksen aihe on toteuttaa suorituskyvyn vertailu yhdistämällä

Apex-luokkia. Apex-ohjelmointikieli mahdollistaa usean eri asynkronisen kontekstin käytön samassa luokassa, kuten ajastetun ja jonotettavan (Marańda, Poniszewska-Maranda ja Szymczyńska 2022). Kontekstien yhdistäminen ja näiden suorituskyvyllinen vertailu olisi uusi näkökulma ja voisi mahdollisesti tuottaa tarkentavaa tietoa.

Kolmas potentiaalinen jatkotutkimusaihe olisi toistaa tutkimus toisenlaisella menetelmällä, esimerkiksi hyödyntäen tilastollisia testejä. Tämä lähestymistapa tarjoaisi mahdollisuuden vertailla ja lisätä tuloksien luotettavuutta, joka voisi tuoda lisävalaistusta suorituskykyerojen tarkempaan ymmärtämiseen. Tällainen monipuolinen lähestymistapa voisi täydentää alkuperäisen tutkimuksen näkökulmaa ja edistää alan kokonaisvaltaista ymmärrystä.

Yhteenvetona voidaan todeta, että tutkimus tarjoaa konkreettisen panoksen Apex-ohjelmointikielellä kehittämiseen ja antaa arvokasta tietoa ohjelmistokehittäjille pyrkiessään optimoimaan suorituskykyä monimutkaisissa sovelluksissa, jotka vaativat serialisointia.

Lähteet

AlJahdali, Hussain, Abdulaziz Albatli, Peter Garraghan, Paul Townend, Lydia Lau ja Jie Xu. 2014. “Multi-tenancy in Cloud Computing”. Teoksessa *2014 IEEE 8th International Symposium on Service Oriented System Engineering,* 344–351. https://doi.org/10.1109/SOSE.2014.50.

Can, Atlas. 2022. “Easily Debug Salesforce Using Nebula Logger”. Haettu 29.10.2023, huhtikuu. https://www.salesforceben.com/easily-debug-salesforce-using-nebula-logger/.

Chaudhary, Amit. 2023. “Wrapper Class in Salesforce”. Haettu 23.20.2023, syyskuu. https://www.apexhours.com/wrapper-class-in-salesforce/.

Ciechan, Damian. 2023. “Comparative analysis of frameworks and automation tools in terms of functionality and performance on the Salesforce CRM Platform”. *Journal of Computer Sciences Institute* 27 (kesäkuu): 154–161. https://doi.org/10.35784/jcsi.3560.

Farrar, Jody. 2023. “What Does Salesforce Do?” Haettu 22.10.2023, syyskuu. https://www.salesforce.com/blog/what-does-salesforce-do/.

Gupta, Prateek. 2022. “Introduction to Imperative Code Execution Machine - a framework for sustainable Salesforce application development”. Teoksessa *2022 IEEE/ACIS 20th International Conference on Software Engineering Research, Management and Applications (SERA),* 30–38. https://doi.org/10.1109/SERA54885.2022.9806725.

Hamerski, Jean Carlo, Anderson R.P. Domingues, Fernando G. Moraes ja Alexandre Amory. 2018. “Evaluating Serialization for a Publish-Subscribe Based Middleware for MPSoCs”. Teoksessa *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS),* 773–776. https://doi.org/10.1109/ICECS.2018.8618003.

Hericko, Marjan, Matjaz B. Juric, Ivan Rozman, Simon Beloglavec ja Ales Zivkovic. 2003. “Object Serialization Analysis and Comparison in Java and .NET”. *SIGPLAN Not.* (New York, NY, USA) 38, numero 8 (elokuu): 44–54. ISSN: 0362-1340. https://doi.org/10.1145/944579.944589.

Hevner, Alan, Alan R, Salvatore March, Salvatore T, Park, Jinsoo Park, Ram ja Sudha. 2004. “Design Science in Information Systems Research”. *Management Information Systems Quarterly* 28 (maaliskuu): 75–105. https://doi.org/https://doi.org/10.2307/25148625.

Krebs, Rouven, Christof Momm ja Samuel Kounev. 2012. “Architectural Concerns in Multi-Tenant SaaS Applications”, 426–431. https://doi.org/10.5220/0003957604260431.

Maeda, Kazuaki. 2012. “Performance evaluation of object serialization libraries in XML, JSON and binary formats”. Teoksessa *2012 Second International Conference on Digital Information and Communication Technology and it’s Applications (DICTAP),* 177–182. h ttps://doi.org/10.1109/DICTAP.2012.6215346.

Manchar, Anuradha, ja Ankit Chouhan. 2017. “Salesforce CRM: A new way of managing customer relationship in cloud environment”. Teoksessa *2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT),* 1–4. https: //doi.org/10.1109/ICECCT.2017.8117887.

Marańda, Witold, Aneta Poniszewska-Maranda ja Małgorzata Szymczyńska. 2022. “Data Processing in Cloud Computing Model on the Example of Salesforce Cloud”. *Information* 13 (helmikuu): 85. https://doi.org/10.3390/info13020085.

Mathew, Reena, ja Ryan Spraetz. 2009. “Test Automation on a SaaS Platform”. Teoksessa *2009 International Conference on Software Testing Verification and Validation,* 317–325. https://doi.org/10.1109/ICST.2009.46.

Nagy, Akos, ja Bence Kovari. 2016. “Analyzing .NET serialization components”. Teoksessa *2016 IEEE 11th International Symposium on Applied Computational Intelligence and Informatics (SACI),* 425–430. https://doi.org/10.1109/SACI.2016.7507414.

Patel, Jigar, ja Ankit Chouhan. 2016. “An approach to introduce basics of Salesforce.com: A cloud service provider”. Teoksessa *2016 International Conference on Communication and Electronics Systems (ICCES),* 1–8. https://doi.org/10.1109/CESYS.2016.7889991.

Salesforce. 2022a. “Execution Governors and Limits”. Haettu 21.20.2023. https://developer. salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_gov_limits.htm.

Salesforce. 2022b. “Platform Multitenant Architecture”. Haettu 21.20.2023. https://architect.salesforce.com/fundamentals/platform-multitenant-architecture.

———. 2023a. “Apex Scheduler”. Versio 59.0, Haettu 29.10.2023. https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_scheduler.htm.

———. 2023b. “Asynchronous Apex”. Versio 59.0, Haettu 29.10.2023. https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_async_overview.htm?q=Asynchronous.

———. 2023c. “Batch Apex”. Versio 59.0, Haettu 29.10.2023. https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_batch.htm.

———. 2023d. “Future Methods”. Versio 59.0, Haettu 29.10.2023. https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_invoking_future_methods.htm.

———. 2023e. “Invoking Apex”. Versio 59.0, Haettu 29.10.2023. https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_invoking.htm.

———. 2023f. “Queueable Apex”. Versio 59.0, Haettu 29.10.2023. https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_queueing_jobs.htm.

———. 2023g. “Salesforce record size overview”. Haettu 29.10.2023, elokuu. https://help.salesforce.com/s/articleView?id=000383664&type=1.

———. 2023h. “Triggers”. Versio 59.0, Haettu 29.10.2023. https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_triggers.htm.

———. 2023i. “Working with sObjects”. Versio 59.0, Haettu 18.11.2023. https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/langCon_apex_SObjects.htm.

Sumaray, Audie, ja S. Kami Makki. 2012. “A Comparison of Data Serialization Formats for Optimal Efficiency on a Mobile Platform”. Teoksessa *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication.* ICUIMC ’12. Kuala Lumpur, Malaysia: Association for Computing Machinery. ISBN: 9781450311724. https://doi.org/10.1145/2184751.2184810.

Walraven, Stefan, Tanguy Monheim, Eddy Truyen ja Wouter Joosen. 2012. “Towards Performance Isolation in Multi-Tenant SaaS Applications”. Teoksessa *Proceedings of the 7th Workshop on Middleware for Next Generation Internet Computing*. MW4NG ’12. Montreal, Quebec, Canada: Association for Computing Machinery. ISBN: 9781450316071. https://doi.org/10.1145/2405178.2405184.

Zaa, Jitendra, ja Anshul Verma. 2016. *Apex Design Patterns*. Packt Publishing. ISBN: 178217365X.

Liitteet

A Asynkronisen Batch Apex-luokan datan lukeminen ja käsittely R:llä

```
library(readr)
library(ggplot2)
library(dplyr)
custom_colors <- c("#FFA100", "#008B8B")

#Listmap
asyncbatch1 <-read.csv("CombinedAsyncListmapBatch.csv", sep = ";")
asyncbatch1$HeapSize_KB <- asyncbatch1$HeapSize / (1024)
summary(asyncbatch1)

#Wrapper
asyncbatch2 <-read.csv("CombinedAsyncWrapperBatch.csv", sep = ";")
asyncbatch2$HeapSize_KB <- asyncbatch2$HeapSize / (1024)
summary(asyncbatch2)

yhdistetty_async_batch_data <- rbind(asyncbatch1, asyncbatch2)

ggplot(data = yhdistetty_async_batch_data, aes(
    x = Tyyppi, y = FullTime)) +
    geom_boxplot(fill = custom_colors) +
    labs(x = "Tyyppi", y = "Kokonaissuoritusaika (ms)")

ggplot(data = yhdistetty_async_batch_data, aes(
    x = Tyyppi, y = CpuTime)) +
    geom_boxplot(fill = custom_colors) +
    labs(x = "Tyyppi", y = "CPU-aika (ms)")
```

```
ggplot(yhdistetty_async_batch_data, aes(
    fill=Tyyppi, y=HeapSize_KB, x=RecordAmount)) +
    geom_bar(position="dodge", stat="identity") +
    scale_x_continuous(breaks = c(250, 500, 750, 1000)) +
    scale_fill_manual(values = custom_colors) +
    labs(x = "Tietue määrä", y = "Kekomuisti (kt)")
```

B Asynkronisen Future Apex-luokan datan lukeminen ja käsittely R:llä

```
library(readr)
library(ggplot2)
library(dplyr)
custom_colors <- c("#FFA100", "#008B8B")

#Listmap
asyncfuture1 <-read.csv("CombinedAsyncListmapFuture.csv",
sep = ";")
asyncfuture1$HeapSize_KB <- asyncfuture1$HeapSize / (1024)
summary(asyncfuture1)
#Wrapper
asyncfuture2 <-read.csv("CombinedAsyncWrapperFuture.csv",
sep = ";")
asyncfuture2$HeapSize_KB <- asyncfuture2$HeapSize / (1024)
summary(asyncfuture2)
yhdistetty_async_future_data <- rbind(
  asyncfuture1, asyncfuture2)
summary(yhdistetty_async_future_data)

ggplot(data = yhdistetty_async_future_data,
  aes(x = Tyyppi, y = FullTime)) +
  geom_boxplot(fill = custom_colors) +
  labs(x = "Tyyppi", y = "Kokonaissuoritusaika (ms)")

ggplot(data = yhdistetty_async_future_data,
  aes(x = Tyyppi, y = CpuTime)) +
  geom_boxplot(fill = custom_colors) +
  labs(x = "Tyyppi", y = "CPU-aika (ms)")

ggplot(yhdistetty_async_future_data,
```

```
aes(fill=Tyyppi, y=HeapSize_KB, x=RecordAmount)) +
geom_bar(position="dodge", stat="identity") +
scale_x_continuous(breaks = c(250, 500, 750, 1000)) +
scale_fill_manual(values = custom_colors) +
labs(x = "Tietue määrä", y = "Kekomuisti (kt)")
```

C Asynkronisen Queueuable Apex-luokan datan lukeminen ja käsittely R:llä

```
library(readr)
library(ggplot2)
library(dplyr)
custom_colors <- c("#FFA100", "#008B8B")

#Listmap queue
asyncQueueable1 <-read.csv("CombinedAsyncListmapQueue.csv",
  sep = ";")
asyncQueueable1$HeapSize_KB <- asyncQueueable1$HeapSize / (1024)
summary(asyncQueueable1)
#Wrapper queue
asyncQueueable2 <-read.csv("CombinedAsyncWrapperQueue.csv",
  sep = ";")
asyncQueueable2$HeapSize_KB <- asyncQueueable2$HeapSize / (1024)
summary(asyncQueueable2)

yhdistetty_async_queueable_data <- rbind(
  asyncQueueable1, asyncQueueable2)
summary(yhdistetty_async_queueable_data)



ggplot(data = yhdistetty_async_queueable_data,
  aes(x = Tyyppi, y = FullTime)) +
  geom_boxplot(fill = custom_colors) +
  labs(x = "Tyyppi", y = "Kokonaissuoritusaika (ms)")

ggplot(data = yhdistetty_async_queueable_data,
  aes(x = Tyyppi, y = CpuTime)) +
```

```
  geom_boxplot(fill = custom_colors) +
  labs(x = "Tyyppi", y = "CPU-aika (ms)")

ggplot(yhdistetty_async_queueable_data,
  aes(fill=Tyyppi, y=HeapSize_KB, x=RecordAmount)) +
  geom_bar(position="dodge", stat="identity") +
  scale_x_continuous(breaks = c(250, 500, 750, 1000)) +
  scale_fill_manual(values = custom_colors) +
  labs(x = "Tietue määrä", y = "Kekomuisti (kt)")
```

D Asynkronisen Schedulable Apex-luokan datan lukeminen ja käsittely R:llä

```
library(readr)
library(ggplot2)
library(dplyr)
custom_colors <- c("#FFA100", "#008B8B")

#Listmap queue
asyncSchedulable1 <-read.csv("CombinedAsyncListmapSchedule.csv",
  sep = ";")
asyncSchedulable1$HeapSize_KB <- asyncSchedulable1$HeapSize / (1024)
summary(asyncSchedulable1)
#Wrapper queue
asyncSchedulable2 <-read.csv("CombinedAsyncWrapperSchedule.csv",
  sep = ";")
asyncSchedulable2$HeapSize_KB <- asyncSchedulable2$HeapSize / (1024)
summary(asyncSchedulable2)

yhdistetty_async_schedule_data <- rbind(
  asyncSchedulable1, asyncSchedulable2)
summary(yhdistetty_async_schedule_data)

ggplot(data = yhdistetty_async_schedule_data,
  aes(x = Tyyppi, y = FullTime)) +
  geom_boxplot(fill = custom_colors) +
  labs(x = "Tyyppi", y = "Kokonaissuoritusaika (ms)")

ggplot(data = yhdistetty_async_schedule_data,
  aes(x = Tyyppi, y = CpuTime)) +
  geom_boxplot(fill = custom_colors) +
```

```
  labs(x = "Tyyppi", y = "CPU-aika (ms)")

ggplot(yhdistetty_async_schedule_data,
  aes(fill=Tyyppi, y=HeapSize_KB, x=RecordAmount)) +
  geom_bar(position="dodge", stat="identity") +
  scale_x_continuous(breaks = c(250, 500, 750, 1000)) +
  scale_fill_manual(values = custom_colors) +
  labs(x = "Tietue määrä", y = "Kekomuisti (kt)")
```

E Synkronisen Apex-luokan datan lukeminen ja käsittely R:llä

```
library(readr)
library(ggplot2)
library(dplyr)
custom_colors <- c("#FFA100", "#008B8B")

sync1 <- read.csv("CombinedSyncListmap.csv", sep = ";")
sync1$HeapSize_KB <- sync1$HeapSize / (1024)
summary(sync1)
#Wrapper
sync2 <- read.csv("CombinedSyncWrapper.csv", sep = ";")
sync2$HeapSize_KB <- sync2$HeapSize / (1024)
summary(sync2)

yhdistetty_sync_data <- rbind(sync1, sync2)
summary(yhdistetty_sync_data)

ggplot(data = yhdistetty_sync_data,
  aes(x = Tyyppi, y = FullTime)) +
  geom_boxplot(fill = custom_colors) +
  labs(x = "Tyyppi", y = "Kokonaissuoritusaika (ms)")

ggplot(data = yhdistetty_sync_data,
  aes(x = Tyyppi, y = CpuTime)) +
  geom_boxplot(fill = custom_colors) +
  labs(x = "Tyyppi", y = "CPU-aika (ms)")

ggplot(yhdistetty_sync_data,
  aes(fill=Tyyppi, y=HeapSize_KB, x=RecordAmount)) +
  geom_bar(position="dodge", stat="identity") +
  scale_x_continuous(breaks = c(250, 500, 750, 1000)) +
```

```
scale_fill_manual(values = custom_colors) +
labs(x = "Tietue määrä", y = "Kekomuisti (kt)")
```

F Kappaleen 4.3 kaaviot R:llä

```
library(readr)
library(ggplot2)
library(dplyr)

asyncbatch1 <-read.csv("CombinedAsyncListmapBatch.csv",
    sep = ";")
asyncbatch1$HeapSize_KB <- asyncbatch1$HeapSize / (1024)
asyncbatch2 <-read.csv("CombinedAsyncWrapperBatch.csv",
    sep = ";")
asyncbatch2$HeapSize_KB <- asyncbatch2$HeapSize / (1024)
asyncfuture1 <-read.csv("CombinedAsyncListmapFuture.csv",
    sep = ";")
asyncfuture1$HeapSize_KB <- asyncfuture1$HeapSize / (1024)
asyncfuture2 <-read.csv("CombinedAsyncWrapperFuture.csv",
    sep = ";")
asyncfuture2$HeapSize_KB <- asyncfuture2$HeapSize / (1024)
asyncQueueable1 <-read.csv("CombinedAsyncListmapQueue.csv",
    sep = ";")
asyncQueueable1$HeapSize_KB <- asyncQueueable1$HeapSize / (1024)
asyncQueueable2 <-read.csv("CombinedAsyncWrapperQueue.csv",
    sep = ";")
asyncQueueable2$HeapSize_KB <- asyncQueueable2$HeapSize / (1024)
asyncSchedulable1 <-read.csv("CombinedAsyncListmapSchedule.csv",
    sep = ";")
asyncSchedulable1$HeapSize_KB <- asyncSchedulable1$HeapSize / (1024)
asyncSchedulable2 <-read.csv("CombinedAsyncWrapperSchedule.csv",
    sep = ";")
asyncSchedulable2$HeapSize_KB <- asyncSchedulable2$HeapSize / (1024)
sync1 <- read.csv("CombinedSyncListmap.csv",
    sep = ";")
```

```
sync1$HeapSize_KB <- sync1$HeapSize / (1024)
sync2 <- read.csv("CombinedSyncWrapper.csv",
    sep = ";")
sync2$HeapSize_KB <- sync2$HeapSize / (1024)

new_colors <- c("#FAD02E", "#86C7D9", "#C7E2A3", "#F5CEB2",
                "#B3A2C7", "#8BD0A1", "#F1A6A0", "#B8B8B8",
                "#A5BDD6", "#FF5733")

yhdistetty_data <- rbind(asyncbatch1, asyncbatch2,
                         asyncfuture1, asyncfuture2,
                         asyncQueueable1, asyncQueueable2,
                         asyncSchedulable1, asyncSchedulable2,
                         sync1, sync2)

ggplot(data = yhdistetty_data,
  aes(x = Tyyppi, y = FullTime, fill = Tyyppi)) +
  geom_boxplot() +
  labs(x = "Tyyppi", y = "Kokonaissuoritusaika (ms)") +
  scale_fill_manual(values = new_colors) +
  theme(legend.position = "right") +
  guides(fill = guide_legend(title = "Tyyppi")) +
  theme(axis.text.x = element_blank())

ggplot(data = yhdistetty_data,
  aes(x = Tyyppi, y = CpuTime, fill = Tyyppi)) +
  geom_boxplot() +
  scale_fill_manual(values = new_colors) +
  labs(x = "Tyyppi", y = "CPU-aika (ms)") +
  theme(legend.position = "right") +
  guides(fill = guide_legend(title = "Tyyppi")) +
```

```
  theme(axis.text.x = element_blank())

ggplot(yhdistetty_data,
  aes(fill=Tyyppi, y=HeapSize_KB, x=RecordAmount)) +
  geom_bar(position="dodge", stat="identity") +
  scale_x_continuous(breaks = c(250, 500, 750, 1000)) +
  scale_fill_manual(values = new_colors) +
  labs(x = "Tietue määrä", y = "Kekomuisti (kt)")
```