Tampereen yliopisto

Kalle Karttunen

VIRTUAALIKONEIDEN SIJOITTELUN OPTIMOINTI PILVIPALVELUISSA ANT COLONY SYSTEM -ALGORITMEILLA

Kandidaattitutkielma
Informaatioteknologian ja viestinnän tiedekunta
Tammikuu 2025

TIIVISTELMÄ

Kalle Karttunen: Virtuaalikoneiden sijoittelun optimointi pilvipalveluissa Ant Colony System - algoritmeilla
Kandidaattitutkielma
Tampereen yliopisto
Informaatioteknologian ja viestinnän tiedekunta, tietotekniikka
Tammikuu 2025

Pilvipalvelujen käyttö on usein halvempaa ja kätevämpää kuin omien tietoteknisten systeemien hankkiminen ja ylläpitäminen, joten on ymmärrettävää, että niiden suosio on kasvanut valtavasti eri organisaatioiden keskuudessa 2010-luvusta eteenpäin. Pilvipalveluja pyöritetään datakeskuksissa, jotka voivat kuluttaa valtavasti energiaa. Ilmastonmuutoksen uhka on luonut painetta kehittää ratkaisuja, joilla tätä energiankulutusta voitaisiin vähentää. Yksi tapa, jolla tämä voidaan saavuttaa ilman tarvetta vaihtaa datakeskuksen fyysisiä komponentteja, on sijoittaa datakeskuksen palvelimilla toimivat virtuaalikoneet niin, että palvelimien resursseja hyödynnetään mahdollisimman tehokkaasti. Tämä sijoitteluongelma on vaikea, ja usein on tyydytty käyttämään varsin yksinkertaisia heuristiikkoja sen ratkaisemiseksi. Sittemmin on kehitetty myös paljon monimutkaisempia menetelmiä, joilla on pyritty saamaan energiatekniikan näkökulmasta parempia tuloksia. Ensimmäinen Ant Colony –optimointimetodi oli Marco Dorigon kehittämä Ant System, joka oli alunperin tarkoitettu kaupparatsuongelman ratkomiseen. Kuusi vuotta myöhemmin julkaistiin Ant Colony System, joka pyrki parantamaan alkuperäisen toteutuksen tehokkuutta. Sittemmin Ant Colony Systemiä on pystytty soveltamaan lukuisiin muihinkin kuin alkuperäiseen ongelmaan, mukaan lukien virtuaalikoneiden sijoitteluongelmaan. Erilaisissa skenaarioissa suoritetuissa testeissä on havaittu, että Ant Colony System –pohjaiset algoritmit tuottavat merkittäviä energiansäästöjä verrattuna suhteessa yksinkertaisiin heuristiikkoihin, ja muihin algoritmeihin.

Tämän työn tarkoituksena on kuvata virtuaalikoneiden sijoitteluongelmaa, sekä esitellä, miten Ant Colony System –pohjaisia algoritmeja on sovellettu siihen kirjallisuudessa. Tämän lisäksi tarkastellaan, ovatko nämä toteutukset onnistuneet saavuttamaan energiasäästöjä, verrattuna vanhempiin ja yksinkertaisempiin sijoitusmetodeihin.

Avainsanat: Pilvipalvelut, Virtualisointi, Algoritmit, Optimointi, Metaheuristiikka

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck –ohjelmalla.

TEKOÄLYN KÄYTTÖ OPINNÄYTTEESSÄ

Opinnäytteessäni on käytetty tekoälysovelluksia:

☐ Ei
☒ Kyllä

Ilmoitukseni mukaan olen käyttänyt opinnäytteessäni tutkielmaprosessin aikana seuraavia tekoälysovelluksia:

Tekoälysovellusten nimet ja versiot:

ChatGPT, Turnitin

Käyttötarkoitus: Tekoälyä on käytetty avuksi lähteiden kielen kääntämiseen, sekä työn alkuperäisyyden tarkastamiseen.

Osiot, joissa tekoälyä on käytetty: Kaikki

Olen tietoinen siitä, että olen täysin vastuussa koko opinnäytteeni sisällöstä, mukaan lukien osat, joissa on hyödynnetty tekoälyä, ja hyväksyn vastuun mahdollisista eettisten ohjeiden rikkomuksista.

ALKUSANAT

Tämä tutkielma on toteutettu vuoden 2024 aikana osana Tampereen yliopiston tietotekniikan kandidaattiohjelmaa.

Haluaisin kiittää muita samaan opinnäytetyöseminaariin osallistuneita, sekä isääni hyödyllisen kritiikin ja neuvojen antamisesta, jotka auttoivat työn valmistumisessa.

Tampereella, 19.1.2025

Kalle Karttunen

SISÄLLYSLUETTELO

1.JOHDANTO 1
2.PILVIPALVELUT 2
2.1 Pilvipalvelujen perusominaisuudet 2
2.2 Pilvipalvelujen hyödyt ja haasteet 3
3.VIRTUAALIKONEET PILVIPALVELUISSA 4
3.1 Virtualisointi 4
3.2 Sijoitusongelma 5
4.ANT COLONY -ALGORITMIT 8
4.1 Metaheuristiikka 8
4.2 Ant Colony –optimointi 9
4.3 Ant Colony System 10
4.4 Virtuaalikoneiden sijoittelu Ant Colony Systemillä 12
5.YHTEENVETO 15
6.LÄHTEET 16

.

1. JOHDANTO

Nykyaikana jatkuvasti kasvava tuotettavan ja jaettavan tiedon määrä on luonut yrityksille haasteita jatkaa liiketoimintaansa omia tietoteknisiä resurssejaan hyödyntäen. Suuret laskentatehovaatimukset aiheuttavat päänvaivaa jo vakiintuneille toimijoille, sekä voivat toimia kynnyksenä estäen uusien yritysten täyden toiminnan. Ratkaisuna tähän ongelmaan ovat syntyneet pilvipalvelut. Ne tarjoavat käyttäjilleen ennennäkemättömät hyödyt esimerkiksi kustannustehokkuudessa, skaalautuvuudessa sekä suorituskyvyssä. On siis ymmärrettävää, että pilvipalvelujen hyödyntäminen on moninkertaistunut Euroopan unionissa sijaitsevissa yrityksissä viimeisen vuosikymmenen aikana [1].

Pilvipalvelut siirtävät IT-resursseihin liittyvät haasteet asiakasyritykseltä palveluntarjoajalle. Yritystoiminnan kannattavuuden ja ympäristönsuojelun näkökulmasta on olennaista, että datakeskuksen virrankulutus ja tietoteknisten resurssien käyttö minimoidaan. Virtuaalikoneet ovat yksi perinteisten pilvipalvelujen keskeisimmistä teknologioista. Valinnat siitä, miten virtuaalikoneet sijoitellaan pilvipalvelua pyörittävän datakeskuksen fyysisille palvelimille, vaikuttavat siihen, miten paljon niiden käyttö vaatii energiaa ja CPU-resursseja. Palveluntarjoajalle tämä valinta ei siis ole triviaali, ja ongelmaan sovelletaan sijoittelualgoritmeja, joilla voidaan pyrkiä minimoimaan virrankulutus tai CPU-resurssien käyttö.

Tämän tutkielman tarkoituksena on tutustuttaa lukija pilvipalvelun käsitteeseen ja sen taustalla piilevään teknologiaan, ja sitten esitellä yksi menetelmä, millä sen energiatehokkuutta voidaan parantaa. Tämä menetelmä on virtuaalikoneiden sijoitusten optimointi Ant Colony System –optimoinnilla. Tutkielma on toteutettu kirjallisuuskatsauksena, ja sen keskeinen tutkimuskysymys on “Miten Ant Colony System –optimointia on hyödynnetty pilvipalvelujen energiatehokkuuden parantamiseen?” Toissijaisena tutkimuskysymyksenä on “Saavutettiinko tällä optimoinnilla paremmat energiasäästöt, kuin muilla sijoitusmetodeilla?”

Luku kaksi sisältää yleisen määritelmän ja katsauksen pilvipalveluihin. Luku kolme kertoo lukijalle oleelliset asiat virtuaalikoneista, sekä niihin liittyvästä sijoitusongelmasta datakeskuksissa. Luku neljä antaa laajemman kuvan Ant Colony –optimoinnista, ja sisältää kirjallisuudesta löytyvien Ant Colony System –toteutusten vertailun. Luku viisi sisältää tutkielman yhteenvedon.

2. PILVIPALVELUT

2.1 Pilvipalvelujen perusominaisuudet

Pilvipalvelu tarkoittaa IT-resurssin, kuten esimerkiksi tallennustilan, palvelimen tai sovelluksen tarjoamista verkon, yleensä internetin välityksellä [2]. Pilvipalvelut toimivat itsepalvelumallilla, jossa asiakas voi käyttää resurssia tarpeen mukaan, ilman tarvetta ihmisten väliseen kontaktiin palveluntarjoajan kanssa. Samalla kun asiakas ostaa palvelun, tehdään tyypillisesti palvelutasosopimus (SLA) (engl. Service level agreement), jossa määritellään tietyt palvelunlaatua koskevat tekniset kriteerit, jotka palveluntarjoaja sitoutuu toteuttamaan. Näitä kriteerejä voivat olla esimerkiksi läpimenoaika tai käytettävyysaika. SLA:ssa sovitaan yleensä myös korvauksesta, jonka palveluntarjoaja on velvollinen maksamaan asiakkaalle, mikäli kriteerit eivät toteudu. [3] Toinen merkittävä ominaisuus on nopea joustavuus, joka tarkoittaa resurssien välitöntä ja usein automaattista skaalaamista käyttäjän tarpeisiin. Resurssien käyttöä myös usein mitataan automaattisesti, mikä mahdollistaa resurssien käytön hallinnan ja siitä raportoimisen. Lisäksi pilvipalvelut ovat tyypillisesti suunniteltu niin, että niitä voi hyödyntää useilla erilaisilla laitteilla, esimerkiksi sekä pöytätietokoneella että matkapuhelimella [2].

Pilvet jaetaan neljään tyyppiin: yksityiseen, julkiseen, yhteisölliseen ja hybridipilveen. Yksityisen pilven resurssit ovat tietyn organisaation yksityiskäytössä. Palvelun tarjoamisen voi kuitenkin hoitaa osittain tai kokonaan kolmannen osapuolen tekijä, eivätkä fyysiset laitteet välttämättä sijaitse organisaation tiloissa [2]. Julkinen pilvi taas on avoinna yleiseen käyttöön, kaikille avoimen internetin välityksellä. Sitä operoi organisaatio, esimerkiksi yksityisyritys tai yliopisto, jonka tiloissa fyysiset laitteet sijaitsevat [2]. Yhteisöllinen pilvi on malli, jossa palvelu on tarkoitettu yksilöille organisaatioista, joilla on yhteisiä tavoitteita tai vaatimuksia. Hybridipilvimalli yhdistää kahden tai useamman edellä mainitun mallin yhteen. Hybridipilvessä kaikki osat säilyvät omina kokonaisuuksinaan, jotka liitetään yhteen standardisoidulla tai patentoidulla teknologialla, joka mahdollistaa tiedonsiirron. [2]

2.2 Pilvipalvelujen hyödyt ja haasteet

Yhä useammat organisaatiot suosivat pilvipalveluja paikallisten IT-resurssien sijasta, niiden lukuisten etujen takia. Niiden merkittävimmät hyödyt liittyvät skaalautuvuuteen, joustavuuteen ja kustannustehokkuuteen. Pilvipalvelun resurssit skaalautuvat käytön mukaan, mikä mahdollistaa uusien instanssien luomisen ja poistamisen ilman tarvetta miettiä resurssien riittävyyttä. Tämä taas mahdollistaa uusien ideoiden testaamisen nopeasti ilman hitaita hankintaprosesseja, mikä tukee organisaation innovaatiota ja nopeuttaa systeemien kehitysprosesseja [4]. Skaalautuvuus parantaa myös organisaation kustannustehokkuutta: Palvelusta maksetaan käytön mukaan, ja käyttöä voidaan aina lisätä tai vähentää tarpeen vaatiessa. Näin vältytään mahdollisilta turhilta kuluilta, joita syntyy käyttämättömistä resursseista. Pilvipalveluita käyttäessä ei myöskään tarvitse palkata henkilöstöä tietoteknisten laitteiden käyttöä tai ylläpitoa varten, pienentäen merkittävästi kuluja.

Monet merkittävät pilvipalvelujen käyttöön liittyvät riskit ja pelot liittyvät tietoturvauhkiin. Tämä johtunee palvelun luonteesta: Viedessä systeemejä tai dataa pilveen, käytännössä luovuttaa ne kolmannen osapuolen laitteiden käsittelyyn. Osa palveluntarjoajista ei edes käytä omia palvelimiaan, vaan halvempia ulkopuolisen toimijan tarjoamia, mikä altistaa asiakkaan entistä suuremmille tietoturvariskeille. Datavarkauksista onkin tullut suuri ongelma pilvipalvelujen tarjoajille. [5] On hyvä myös huomioida, että asiakkaan kokema tietoturva voi toki myös parantua, kun löytyy luotettava palveluntarjoaja, jolla on riittävä tietoturvaosaaminen.

Toinen yleinen pelko on omien ohjelmistojen ja datan saatavuus [5]. Jos palvelimet ovat tukossa, esimerkiksi korkean käyttöasteen tai palvelunestohyökkäyksen seurauksena, niin asiakas ei välttämättä pääse käyttämään niitä ollenkaan. Sama voi tapahtua, jos esimerkiksi tietoverkoissa on häiriöitä, eikä asiakas pysty luomaan yhtyettä palveluun.

3. VIRTUAALIKONEET PILVIPALVELUISSA

Yksi keskeisimmistä pilvipalvelujen toiminnan mahdollistavista teknologioista on virtualisointi [6]. Tässä luvussa käsitellään yhtä virtualisoinnin osa-aluetta, virtuaalikoneita, ja esitellään niihin liittyvä sijoitusongelma, joka kohdataan pilvipalveluja tarjottaessa.

3.1 Virtualisointi

Tietotekniikassa virtualisointi tarkoittaa jonkin fyysisen komponentin muuttamista virtuaaliseksi, loogiseksi komponentiksi [7]. Kun kokonainen tietokone virtualisoidaan, lopputulosta kutsutaan virtuaalikoneeksi. Virtuaalikone toimii tismalleen samalla tavalla kuin fyysinen tietokone, sisältäen käyttöjärjestelmän, sekä tietokoneen muodostavien komponenttien, kuten suorittimen, muistin ja verkkoyhteyksien virtualisoidut korvikkeet. Nämä komponentit määritellään erillisessä konfiguraatiotiedostossa, joka luodaan virtuaalikoneen luomisen yhteydessä. [7] Virtuaalikoneiden tuoma hyöty on se, että niitä voidaan luoda yhdelle fyysiselle laitteelle useita, mikä mahdollistaa useiden erilaisten systeemien käyttämisen ilman tarvetta hankkia useampaa kuin yhtä fyysistä systeemiä.

Virtuaalikoneiden pyörittäminen vaatii erillisen, virtualisoidun ja fyysisen laitteen eristävän ohjelmiston käyttöä. Tällaista ohjelmistoa kutsutaan virtualisointialustaksi (engl. *hypervisor*). Virtualisointialustan alkuperäinen ja ehkä keskeisin tehtävä on resurssien hallinta ja jakaminen virtuaalikoneille. [7] Virtualisointialusta sijoittuu siis virtuaalikoneiden ja fyysisen laitteiston väliin, ja toimii eräänlaisena fyysisten resurssien välittäjänä, joka kommunikoi molempien puolien kanssa. Ilman virtualisointialustaa virtuaalikoneet yrittäisivät kaikki hallita samaa fyysistä laitteistoa, ilman minkäänlaista koordinaatiota tai tietoa toisistaan, mikä johtaisi helposti kaaokseen ja ristiriitoihin. Resurssien jakamista varten virtualisointialusta luo jokaiselle virtuaalikoneelle mukautetun kuvan sen käytössä olevista fyysisistä resursseista. Käytännössä tämä tarkoittaa sitä, että virtualisointialusta uskottelee sen alaisuudessa oleville virtuaalikoneille, että ne ovat tekemisissä suoraan fyysisen laitteiston kanssa, vaikka todellisuudessa niiden tekemät pyynnöt menevät virtualisointialustan käsittelyyn. Usein tähän sisältyy myös se, että virtualisointialustan esittämät komponentit eivät vastaa todellista fyysistä laitetta. [7] Virtuaalikone saattaa esimerkiksi uskoa kirjoittavansa 250 gigatavun muistilevylle, vaikka todellisuudessa fyysistä muistia on käytössä teratavun verran. Näin varmistetaan, että virtuaalikoneet eivät käytä enempää resursseja kuin niille on määrätty.

Virtualisointialustat jaetaan yleisesti kahteen tyyppiin, tyypin 1 ja tyypin 2 virtualisointialustoihin. Tyyppi 1 on suoraan yhteydessä fyysiseen laitteistoon, ja sitä kutsutaankin

välillä bare-metal -toteutukseksi. [7] Tyyppi 2 taas toimii erillisen käyttöjärjestelmän päällä, jonka kautta tieto kulkee virtualisointialustan ja fyysisen laitteiston välillä. Molemmissa lähestymistavoissa on omat hyvät ja huonot puolensa. Tyyppi 2 soveltuu paremmin yksittäisen kehittäjän käytettäväksi hänen henkilökohtaisella tietokoneellaan, koska se on tyypillisesti helppo asentaa ja ottaa käyttöön, ja koska sen käyttö ei estä muiden ohjelmien samanaikaista käyttöä samalla tietokoneella. [7] Tyyppi 1 soveltuu taas paremmin palvelinkäyttöön useista syistä. Tyypin 1 virtualisointialustat ovat yleisesti tehokkaampia, koska ne pääsevät kommunikoimaan suoraan alla olevan laitteiston kanssa. Ne ovat myös turvallisempia, koska niissä virtuaalikoneiden prosessit ovat täysin eristettyjä, eivätkä ne pääse vaikuttamaan virtualisointialustaan tai muihin virtuaalikoneisiin. Tämän lisäksi tyypin 1 toteutukset vaativat niille omistetun laitteiston, joilla ei voida ajaa muita ohjelmia samanaikaisesti. [7]

Yksi yleisistä pilvipalvelujen tarjoamista virtualisointipalveluista on palvelinvirtualisointi [8]. Palvelinvirtualisointi voidaan vielä jakaa useampaan alalajiin, mutta tämän työn kannalta olennainen on niin kutsuttu täysvirtualisointi, jossa kokonainen fyysinen systeemi virtualisoidaan, eli luodaan virtuaalikone, jolla voidaan ajaa muokkaamattomia käyttöjärjestelmiä [8]. Suosittuja täysvirtualisointipalveluja ovat esimerkiksi VMvare ESXi, Microsoft Hyper-V ja KVM (Kernel-based Virtual Machine), jotka kaikki toimivat tyypin 1 virtualisointialustoilla.

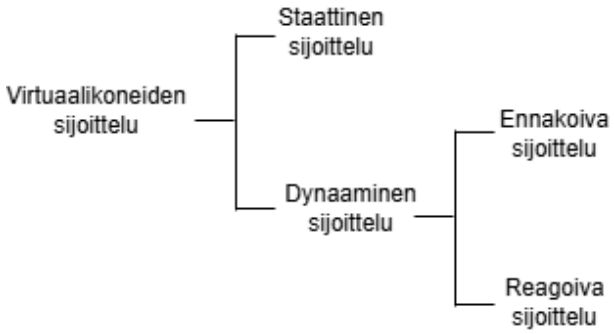
3.2 Sijoitusongelma

Virtualisointiteknologia on pilvipalvelujen kulmakivi [6]. Palveluntarjoajan on kuitenkin kiinnitettävä huomiota siihen, miten virtuaalikoneet sijoitetaan käytössä oleville palvelimille useista eri syistä. Tilannetta, jossa palvelimella on liikaa virtuaalikoneita, kutsutaan ylikuormitukseksi. Ylikuormitus heikentää palvelimen suorituskykyä, ja saattaa aiheuttaa SLA-rikkomuksia, kun ylikuormitettu palvelin ei pysty tarjoamaan asiakkaalle luvattua kapasiteettiä tai käytettävyysaikaa. [9] [10] Ylikuormitus heikentää myös datakeskuksen energiatehokkuutta. Täyteen lastatut palvelimet suoriutuvat tehtävistään hitaammin, kasvattaen niiden käyttöaikaa, ja ne vaativat lisäksi enemmän jäähdytystä. [10] Toisaalta palvelimen alikuormitus ei myöskään ole ihanteellinen tilanne. Jos virtuaalikoneet jaotellaan liian usealle palvelimelle, niin että jokaisella palvelimella on vain pieni määrä virtuaalikoneita, niin datakeskuksen energiatehokkuus jälleen kerran huononee. [9] Näin ollen ideaalinen tilanne on se, että virtuaalikoneet on sijoitettu mahdollisimman pienelle

määrälle palvelimia, kuitenkin niin, että ylikuormitusta ei tapahdu. Ihanteellisella sijoittelulla on muitakin hyötyjä, kuten esimerkiksi virtuaalikoneiden välisen kommunikoinnin tehostaminen [10]. Tässä työssä kuitenkin keskitytään energiatehokkuuden maksimointiin pyrkivään sijoitteluun.

Yksi tapa toteuttaa virtuaalikoneiden sijoittelu on niin kutsuttu staattinen sijoittelu. Staattisessa sijoituksessa virtuaalikoneen sijainti määrätään kerran, ja se säilyy samana koko sen elinkaaren ajan. [9] Tätä lähestymistapaa hyödynnetään yleensä esimerkiksi järjestelmän käynnistyksen yhteydessä, kun kaikki virtuaalikoneet on sammutettava, uudelleenkäynnistettävä ja uudelleensijoitettava, tai kun halutaan lisätä uusi virtuaalikone [11]. Staattisella lähestymistavalla on kuitenkin omat rajoituksensa, jotka tekevät siitä useissa tilanteissa tehottoman ratkaisun. Ehkä suurin näistä on se, että staattinen ratkaisu ei mahdollista mukautumista muuttuviin, ja vaikeasti ennustettaviin asiakkaiden resurssivaatimuksiin. [11]

Käytännössä resurssien käyttö on siis usein muuttuvaa ja vaikeasti ennustettavaa, mikä luo tarpeen virtuaalikoneiden mahdolliselle uudelleensijoittamiselle. Tämän mahdollistaa dynaaminen sijoittelu. Dynaamisessa sijoittelussa virtuaalikoneiden sijainti voi muuttua niiden elinkaaren aikana, eli niitä voidaan siirtää palvelimien välillä. [9] Tällainen sijoittelu voivaan tehdä joko reaktiivisesti tai ennakoivasti. Reaktiivisessa lähestymistavassa systeemin tilaa seurataan, ja siirrot suoritetaan, kun jokin ennalta määrätty ehto täyttyy. Tämä ehto voi olla esimerkiksi suorituskyvyn lasku tietyn tason alapuolelle. Ennakoivassa sijoittelussa taas pyritään suorittamaan sijoitus ennen kuin tällainen ehto täyttyy. [9]



Kuva 1. *Sijoittelumenetelmien luokittelu. [9]*

Optimaalisen sijoittelun löytäminen suurissa datakeskuksissa, joissa on lukuisia palvelimia ja paljon virtuaalikoneita, voi olla erittäin monimutkaista. Tämä muun muassa siksi, että jos fyysisten palvelimien lukumäärä on m, ja virtuaalikoneiden n, niin mahdollisten sijoitusratkaisujen lukumäärä voi olla m^n. [9] Näin ollen ihanteellisten ratkaisujen löytämiseksi on kehitetty erilaisia menetelmiä. Tällaiset menetelmät koostuvat yleensä kolmesta askeleesta: Ensin on määritettävä, mitkä virtuaalikoneet halutaan siirtää. Sitten

on määrättävä minne ne pitäisi siirtää. Lopuksi on vielä määriteltävä millä ajanhetkellä siirto kannattaa suorittaa. [10] Kaikilla näillä valinnoilla on merkitystä datakeskuksen toiminnan kannalta. Siirron kohteeksi tulevan virtuaalikoneen valinta sekä siirron päämääränä olevan palvelimen valinta voivat perustua esimerkiksi virrankulutuksen minimoimiseen tai katkosten välttämiseen ja näin ollen SLA-rikkomusten minimoimiseen [10]. Myös siirron ajankohdalla on merkitystä, koska se on tyypillisesti resurssienkulutuksen kannalta intensiivinen operaatio, joka saattaa aiheuttaa huomattavia häiriöitä palvelunlaadussa huonosti ajoitettuna [10].

Optimaalisten siirtojen löytämistä käsitellään usein kombinatorisena optimointiongelmana [10]. Mahdollisten ratkaisujen suuren lukumäärän vuoksi ihanteellisen vaihtoehdon määrittäminen suoraviivaisesti tarkistamalla kaikki mahdollisuudet ei yleensä ole käytännöllinen lähestymistapa ongelmaan. Näin ollen on käytetty lukuisia algoritmeja ja heuristiikkoja optimaalisen tai ainakin lähes optimaalisen ratkaisun löytämiseen. Ne voivat painottaa lukuisia eri asioita, kuten esimerkiksi energiatehokkuuden maksimointia, kustannusten minimoimista tai vaikka siirtojen määrän minimoimista. [10, 11] Tässä työssä keskitytään käsittelemään energiatehokkuuteen tähtääviä ratkaisuja, sillä niitä tuntuu esiintyvän kirjallisuudessa eniten, ja koska energiankulutuksen vähentäminen on hyvin olennainen osa kestävää kehitystä ja ilmastonmuutoksen torjuntaa.

4. ANT COLONY -ALGORITMIT

Tässä luvussa käsitellään Ant Colony –optimointiin perustuvia sijoitusalgoritmeja. Luvun alussa perehdytään lyhyesti ja pintapuolisesti kombinatorisen optimoinnin ja metaheuristiikan käsitteisiin, jonka jälkeen siirrytään Ant Colony –optimoinnin pariin. Lopulta esitellään ja vertaillaan kolmea suhteellisen uutta kirjallisuudesta löytynyttä Ant Colony System –pohjaista sijoitusalgoritmia.

4.1 Metaheuristiikka

Kombinatorinen optimointi on matematiikan osa-alue, joka keskittyy "optimaalisen" ratkaisun löytämiseen ääreellisen kokoisesta ratkaisujoukosta [12]. Kombinatorinen optimointiongelma $P = (S, f)$ koostuu seuraavista osista:

- Muuttujajoukosta $X = \{x_1, \ldots, x_n\}$,
- muuttujien määrittelyjoukoista $M_1, \ldots, M_n$,
- muuttujiin kohdistuvista rajoitteista,
- kohdefunktiosta f :

  $f: M_1 \times \cdots \times M_2 \rightarrow \mathbb{R}^+$,
- Sekä joukosta S , joka sisältää kaikki sallitut ratkaisut:

  $S = \{s = \{(x_1, y_1), (x_n, y_n)\} | \, y_i \in M_i \, , ja\, s\, noudattaa\, rajoitteita\}$ [12]

Joukkoa S kutsutaan usein ratkaisualueeksi (engl. *Solution space*). Kombinatorisen optimoinnin tavoitteena on löytää jokin ratkaisualueeseen kuuluva alkio, joka minimoi kohdefunktion arvon. [12] Tutkittuja kombinatorisia optimointiongelmia ovat esimerkiksi kauppamatkustajan ongelma (engl. *Travelling salesman problem*), reppuongelma (engl. *Knapsack problem*), sekä virtuaalikoneiden optimaalisen sijoittelun löytäminen datakeskuksissa.

Ratkaisualueen ollessa suuri, parhaan ratkaisun etsiminen voi kuitenkin olla niin hidasta, ettei se aina ole käytännössä kannattavaa. Esimerkiksi virtuaalikoneiden sijoitteluongelman on todettu olevan vaativusluokaltaan NP-kova. [9 11] Käytännössä tämä tarkoittaa muun muassa sitä, että parhaan mahdollisen ratkaisun löytämiseen kuluva aika voi pahimmassa tapauksessa kasvaa eksponentiaalisesti [12]. Usein on siis käytännöllistä tyytyä riittävän hyvään mutta ei välttämättä täydelliseen ratkaisuun. Yksi keino tällaisten ratkaisujen tuottamiseen on soveltaa heuristisia metodeja. Heuristiikkaa hyödyntävät metodit ovat suunniteltu juuri tuottamaan kelvollisia ratkaisuja hyväksyttävässä ajassa,

mutta ne ovat hyvin ongelmakohtaisia, eivätkä yksittäiset toteutukset välttämättä siis ole yleisesti hyödyllisiä. [12]

Myöhemmin kuitenkin on kehitetty uudenlaisia algoritmimalleja, jotka pystyvät soveltamaan heuristiikkaa yleisellä tasolla mihin tahansa optimointiongelmaan. Tällaisia algoritmeja kutsutaan metaheuristisiksi algoritmeiksi [13]. Metaheuristiikalle ei ole olemassa yhtä sovittua määritelmää. [12] Sen perusideana on kuitenkin toimia korkeamman tason proseduurina, jonka tehtävänä on ohjata sen alla toimivia heuristiikkoja löytämään ratkaisuja. Tähän se hyödyntää ymmärrystä tehtäväkohtaisesta ratkaisualueesta, mikä mahdollistaa saman algoritmin soveltamisen moniin erilaisiin ongelmiin. [12, 13] Toisistaan eroavia metaheuristisia algoritmeja on kehitetty useita 1970-luvulta lähtien, ja niitä voidaan luokitella erilaisten ominaisuuksien perusteella [12, 13]. Yksi tällainen ominaisuus on se inspiraation lähde, minkä pohjalta algoritmi on alun perin kehitetty. Tästä näkökulmasta katsottuna metaheuristiikat voidaan jakaa karkeasti kahteen eri luokkaan: luontoon perustuviin ja muihin algoritmeihin. Luonnon inspiroimat algoritmit mukailevat jotain luonnossa esiintyvää mekanismia, tai jonkin eläinlajin tyypillistä käyttäytymistä. Esimerkkejä luonnonilmiöihin perustuvista metaheuristiikoista ovat esimerkiksi geneettinen algoritmi, joka perustuu Charles Darwinin kehittämään evoluutioteoriaan ja luonnonvalintaan, Ant Colony –optimisaatio, joka perustuu muurahaisten käyttäytymiseen ja niiden väliseen feromonikommunikointiin, sekä Particle Swarm –optimisaatio, joka perustuu lintu- tai kalaparvien sosiaaliseen käyttäytymiseen. [12, 13]

4.2 Ant Colony –optimointi

Kun muurahaisyhdyskunta löytää ympäristöstään uuden ravintolähteen, yksittäiset muurahaiset alkavat vaeltaa sen ja kekonsa välillä. Aluksi jokainen muurahainen muodostaa reittinsä itsenäisesti ja varsin sattumanvaraisesti. Ajan kuluessa kuitenkin nopeimmat reitit ravinnon ja keon välillä yleistyvät, ja hitaampien reittien käyttö vähentyy. Tämä ilmiö on pystytty selittämään muurahaisten välisellä feromonikommunikoinnilla. [14] Kun muurahainen kulkee valitsemallaan reitillä, se laskee maahan feromonia, jonka muut yksilöt aistivat. Lyhyempää reittiä kulkevat muurahaiset ehtivät kulkea kekonsa ja ravinnonlähteen välillä useammin samassa ajassa kuin pidempää reittiä käyttävät. Näin nopeimman reitin feromonikonsentraatio nousee korkeammaksi kuin muiden. Muurahaiset suosivat reittejä, joilla on enemmän feromonia, joten yhä useampi yksilö alkaa ajan kuluessa käyttää nopeampaa reittiä, mikä nostaa sen feromonikonsentraatiota ja suosiota entisestään.

Feromonit myös haihtuvat, mikä saa hitaammat ja käyttämättömät reitit unohtumaan ajan saatossa. [14]

Tähän luonnonilmiöön perustuen on kehitetty Ant Colony –optimointi (ACO). ACO:n perustuen on kehitetty lukuisia algoritmeja, mutta niillä kaikilla on samat perusominaisuudet. [14] ACO-algoritmeissa ongelma mallinnetaan yhtenäisenä graafina, jota pitkin simuloidut muurahaiset kulkevat. Tavoitteena on tuottaa jonkinlainen optimaalinen reitti graafissa. Ratkaisu saavutetaan iteratiivisesti: jokaisella toistokerralla tietty määrä simuloituja muurahaisia kulkee graafin kaaria pitkin solmusta toiseen sattumanvaraisesti ja muodostaa oman reittinsä. Ratkaisun parantamiseksi iteraatioiden välillä sovelletaan feromonisysteemiä. Graafin jokaiselle kaarelle annetaan oma feromoniarvonsa, joka vaikuttaa siihen, kuinka todennäköisesti muurahainen päättää jatkaa polkuaan kulkemalla kyseistä kaarta pitkin. Jokaisen iteraation päätteeksi päivitetään kaikkien reittien feromoniarvot, jotka voivat joko kasvaa tai kutistua. Feromonisysteemin tarkempi toteutus on tapauskohtainen, riippuen siitä, millaisia ratkaisuja halutaan tuottaa. Sen tarkoituksena on kuitenkin aina saada muurahaiset suosimaan suotuisia ratkaisuja. [14]

ACO on formalisoitu myös metaheuristiikaksi, mikä mahdollistaa sen soveltamisen yleisellä tasolla moniin erilaisiin kombinatorisiin optimointiongelmiin [14]. Aluksi on muodostettava ongelmaa kuvaava rakennegraafi, joka saadaan esittämällä ratkaisukomponenttien joukko joko graafin solmuina tai kaarina. Tässä yhteydessä ratkaisukomponenttien joukko viittaa joukkoon, joka sisältää kaikki sallitut muuttujien arvojen määräämiset, eli toisin sanoen kaikki sallitut muuttuja-arvo parit. Kun rakennegraafi on selvillä, voidaan siihen soveltaa ACO:ta. [14] Alla on esitetty ACO-heuristiikan algoritmi.

```
1  Aseta parametrit, sekä feromonien alkuarvot
2  kun lopetusehto ei ole täyttynyt toista:
3        määritäMuurahaisReitit
4        suoritaPaikallinenHaku //vapaaehtoinen vaihe
5        päivitäFeromoniarvot
6  lopetus
```

Algoritmi 1. *ACO-metaheuristiikka esitettynä pseudokoodilla. [14]*

Algoritmissa 1 esiintyvä paikallinen haku on heuristiikka, jota sovelletaan tapauskohtaisesti muurahaisten tuottamien ratkaisujen parantamiseen, ennen feromonien päivittämistä. Se ei ole pakollinen osa algoritmia. [14]

4.3 Ant Colony System

Ensimmäinen ACO-algoritmi on nimeltään Ant System, ja sen julkaisi tohtori Marco Dorigo kollegoineen vuonna 1991 [14]. Siinä muurahaiset valitsevat reittinsä ottaen huomioon sekä kaarien feromoniarvot että pituudet, suosien lyhyempiä kaaria. Kaarien fero-

moniarvot päivitetään jokaisen muurahaisen kulkemalle reitille jokaisen iteraation lopuksi. Se kykenee tuottamaan käyttökelpoisia ratkaisuja hyväksyttävässä ajassa, mutta ainoastaan graafin koon ollessa pieni. Esimerkiksi kauppamatkustajan ongelmaa ratkoessa sen suoritusaikojen tiedetään kasvavan liian suureksi, kun graafin koko ylittää 30 solmua. [15] Näin ollen kehitettiin uusi algoritmi, Ant Colony System (ACS), joka pyrkii korjaamaan alkuperäisen Ant Systemin tehokkuusongelmat. Se muokkaa alkuperäisen mallin muurahaisten käytöstä uudella tilansiirtosäännöllä: Reittien muodostus ei keskity enää joka kerta tunnettuja polkuja pitkin kulkemiseen, vaan voidaan myös painottaa uusien, vähemmän kuljettujen reittien tutkimusta. Valinta siitä, kumpaa lähestymistapaa muurahaiset käyttävät, määritetään arpomalla satunnaisluku *u* väliltä [0...1], ja vertaamalla sitä algoritmin käyttäjän itse asettamaan parametriin *v*, jonka on myös oltava väliltä [0...1]. *v*:n arvoa muuttamalla voidaan näin vaikuttaa muurahaisten käytökseen. Tällaista reitinmuodostusta kutsutaan näennäissatunnaiseksi liikkumissäännöksi. Myös feromonisysteemiin tehtiin muutoksia: alkuperäisessä mallissa iteraation lopuksi tehtävä feromonipäivitys suoritetaan nyt ainoastaan parhaalle löydetylle reitille, ja tätä päivitystä kutsutaan ACS:ssä globaaliksi päivitykseksi. [15] Globaalin lisäksi luotiin myös uusi, paikallinen feromonipäivitys, jonka jokainen yksittäinen muurahainen tekee kulkemalleen reitille. Paikallinen päivitys pienentää kuljetun reitin feromoniarvoja, ja sen tarkoituksena on saada muurahaiset tuottamaan monipuolisempia ratkaisuja yhden iteraation aikana: kun muurahainen reitin kuljettuaan vähentää sen feromoniarvoja, seuraavat yksilöt käyttävät pienemmällä todennäköisyydellä samaa reittiä. [14 15] Taulukossa 1 listataan vielä selkeyden vuoksi ACS:in ja alkuperäisen Ant System –algoritmin perusominaisuudet.

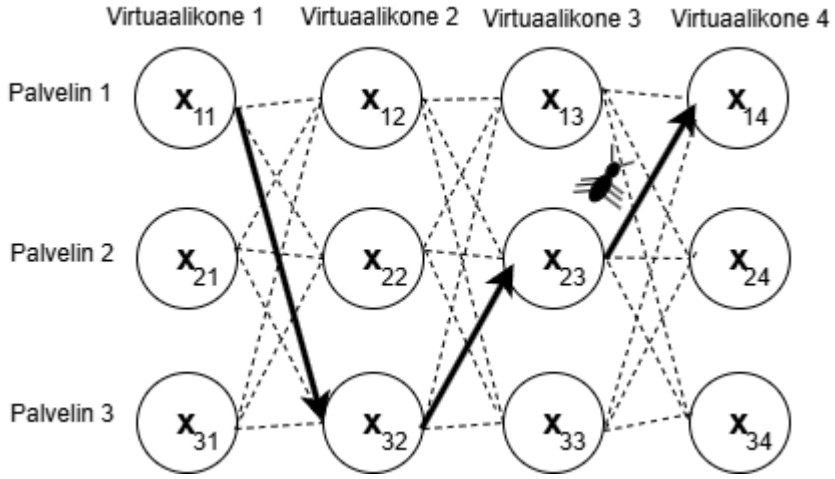
Algoritmi	Ant System	Ant Colony System
Julkaisuvuosi	1991	1996
Reitinmuodostus	Perustuu aina kaarien feromoniarvoihin ja pituuksiin: Suosii tunnettuja ja lyhyitä reittejä.	Käyttää näennäissatunnaista liikkumissääntöä: Voi suosia tunnettuja ja lyhyitä reittejä, tai vaihtoehtoisesti vähemmän kuljettujen polkujen tutkimista.
Feromonipäivitys	Suoritetaan iteraation lopuksi jokaisen muurahaisen muodostamalle reitille. Kuljettujen polkujen feromoniarvoja vahvistetaan.	Iteraation lopuksi ainoastaan lupaavimman löydetyn polun feromoniarvoja vahvistetaan. Kuljettuaan kaaren yli yksittäiset muurahaiset suorittavat paikallisen feromonipäivityksen, joka laskee sen feromoniarvoa.

Taulukko 1. Ant systemin ja Ant Colony Systemin perusominaisuudet. [14 15]

4.4 Virtuaalikoneiden sijoittelu Ant Colony Systemillä

Ant Colony Systemiä on sovellettu jonkin verran virtuaalikoneiden sijoitteluongelman ratkaisuun [16]. Tässä alaluvussa esitellään lyhyesti kolme kirjallisuudessa esitettyä esimerkkitoteutusta, sekä vertaillaan niillä saatuja kokeellisia energiansäästötuloksia. Valikoituneet algoritmit ovat Gaon C et al. julkaisema toteutus vuodelta 2016, Liun X et. al julkaisema toteutus vuodelta 2018, sekä Alharbin F et al. julkaisema toteutus vuodelta 2019.

Jokaisessa valitussa julkaisussa sijoitusongelmaa kuvaava graafi esitetään vierusmatriisina. Tämä vaikuttaa olevan tavanomainen tapa aihetta käsittelevissä julkaisuissa. [16, 17, 18] Vierusmatriisi koostuu muuttujista, joilla esitetään kaikki mahdolliset sijoitukset eli virtuaalikone-palvelin parit. Muuttujien arvot määräytyvät seuraavasti: Jos virtuaalikone i on sijoitettu palvelimelle j , niin sijoitusta kuvaava muuttuja $x_{ij} = 1$, muussa tapauksessa $x_{ij} = 0$. Jokainen vierusmatriisin muuttuja vastaa yhtä solmua graafissa. Muurahaiset rakentavat ratkaisunsa kulkemalla solmujen läpi, kunnes jokainen virtuaalikone on sijoitettu jollekin palvelimelle. Kuvasta 2 selviää, miten ratkaisun rakentamisprosessi visualisoitiin Liun X et al. julkaisussa.



Kuva 2. *Esimerkki muurahaisyksilön muodostamasta ratkaisusta, kun palvelimia on käytössä 3, ja sijoitettavia virtuaalikoneita on 4. Muurahaisen reitti graafissa on merkitty paksuilla nuolilla. [17]*

Kuvassa 2 siis on muodostettu ratkaisu, jossa palvelin 1 saa virtuaalikoneet 1 ja 4, palvelin 2 saa virtuaalikoneen 3, ja palvelin 3 saa virtuaalikoneen 2. Kaikille yhteistä on myös feromonipäivitykset: globaalit ja paikalliset päivitykset olivat toteutettu aivan samalla tavalla kuin ne oli kuvattu alkuperäisessä ACS-julkaisussa. Myös muurahaisten reitinvalinta toimi identtisesti, sillä kaikissa toteutuksissa muurahaiset noudattavat samaa, alkuperäistä näennäissatunnaista liikkumissääntöä [16, 17, 18].

Yksi merkittävimmistä eroista algoritmien välillä piilee siinä, miten niissä on asetettu feromonien alkuarvot. Feromoneja on asetettu kahdella tavalla: joko virtuaalikoneen ja palvelimen välille tai vaihtoehtoisesti kahden virtuaalikoneen välille. Jälkimmäisen mukaan toimitaan Liun et al. julkaisussa, ja edellisen mukaan muissa. [16, 17, 18] Feromonin tallennus virtuaalikoneen ja palvelimen välille tuntuu intuitiivisemmalta, jonka myöntävät Liu. X et al. itsekin. He kuitenkin perustelevat valintaansa sillä, että jos datakeskuksessa on useita identtisiä palvelimia, niin pelkästään palvelimien ominaisuuksia tarkastelemalla ei pystytä päättämään, mikä olisi paras kohde sijoitukselle. Heidän mukaansa olennaista on sen sijaan katsoa, mitä virtuaalikoneita on sijoitettu samoille palvelimille. Näin ollen he päätyivät ratkaisuun, joka keskittyy siihen, miten virtuaalikoneet ryhmittyvät keskenään. [17] Heidän algoritmissaan feromonien alkuarvo eli se feromoniarvo, joka kaarille laitetaan ennen ensimmäistä iteraatiota, on käytettävissä olevien palvelimien lukumäärän käänteisluku [17]. Caon et al. julkaisussa ei ollut mainittu ollenkaan, mikä feromonien alkuarvoksi asetetaan [18]. Alkuarvon voidaan kuitenkin olettaa olevan muurahaisen reitin (eli ratkaisun) pituuden ja graafin solmujen lukumäärän tulon käänteisluku, koska tätä arvoa käytettiin alkuperäisessä ACS-julkaisussa [15]. Alharbi, F et al. taas määrittävät alkuarvot First-fit-decreasing-heuristiikan (FFD) tuottaman, alhaisen energiakulutuksen saavuttaneen ratkaisun perusteella. Ideana on tehostaa muurahaisten etsintäprosessia ja näin tuottaa ratkaisuja nopeammin. [16]

Toinen olennainen ero toteutusten välillä on se, mitä heuristiikkoja niissä sovelletaan. Jokainen algoritmi tavoittelee datakeskuksen energiatehokkuuden parantamista, mutta lähestymistavat tähän eroavat hieman toisistaan [16, 17, 18]. Liu, X et al. suunnittelivat heuristiikan, joka pyrkii tuottamaan samanaikaisesti sekä palvelimia tehokkaasti hyödyntäviä että niiden resursseja tasaisesti kuluttavia ratkaisuja. Käytännössä tämä tarkoittaa sitä, että sijoitukset pyritään tekemään niin, että käynnissä olevia palvelimia olisi mahdollisimman vähän ja että niiden resursseja, eli prosessointitehoa ja muistia, käytettäisiin mahdollisimman tasapuolisesti. Tämä siksi, että muutaman palvelimen käytön korkealla teholla tiedetään olevan energiatehokkaampaa kuin useamman palvelimen käyttö matalammalla teholla. [17] Alharbi, F et al. esittelivät heuristiikan, joka myös tähtää palvelimien tehokkaaseen hyödyntämiseen. Siinä otetaan huomioon sekä palvelimen energiatehokkuus, että sen suorittimen käyttöaste ja kapasiteetti. Tämän lisäksi heidän toteutuksessaan luodaan sekä virtuaalikoneille että palvelimille omat profiilit avustamaan päätöksentekoa. Virtuaalikoneiden profiilit sisältävät tiedot esimerkiksi niiden muisti- ja suoritintehokapasiteettivaatimuksista sekä odotetusta elinajasta. Palvelimien profiilit luodaan perustuen aiemmin kerättyyn lokitietoon, ja ne sisältävät tietoa esimerkiksi palvelimen kapasiteettien tyypillisestä käytöstä, sekä sen energiakulutuksen minimi- ja maksi-

miarvoista. Profiilien ideana on tuottaa lisää hyödyllistä tietoa ja nopeuttaa energiatehokkaiden sijoituspäätösten luomista. [16] Cao, C et al. ottavat huomioon myös tietoverkkojen vaikutuksen palvelimien energiatehokkaaseen käyttöön. Mitä lähempänä toisiaan virtuaalikoneet ovat verkossa, sitä vähemmän resursseja niiden välinen tietoliikenne kuluttaa. Näin ollen tekijät loivat heuristiikan, joka pyrkii asettamaan intensiivisesti keskenään kommunikoivat virtuaalikoneet mahdollisimman lähelle toisiaan tietoverkossa, ideaalisesti samalle palvelimelle. Tämän lisäksi käytetään kahta muutakin, hieman tavanomaisempaa heuristiikkaa: Yhtä, joka painottaa energiatehokkaita sijoituksia, ja toista, joka ohjaa sijoituksia niin, että palvelimien käyttöaste nousee, mikä myös lopulta johtaa parempaan energiahyötysuhteeseen datakeskuksessa. [18]

Algoritmeilla saavutettujen kokeellisten tulosten suora vertailu keskenään olisi vähintäänkin hyvin haastavaa, koska koejärjestelyt erosivat merkittävästi eri julkaisujen välillä. Palvelimet, virtuaalikoneet sekä tietokoneet, joilla varsinaista algoritmia käytettiin, olivat erilaiset jokaisessa testissä. Tämän lisäksi ACS sisältää ohjelmoijan itse asettamia parametreja, jotka vaikuttavat merkittävästi siihen, miten hyvin algoritmi suoriutuu. Nämä parametrit määritettiin eri tavalla jokaisessa testissä. [16, 17, 18] Liu, X et al. vertasivat algoritmiaan lukuisiin erilaisiin heuristiikkoihin, kuten esimerkiksi alkuperäiseen ACO-algoritmiin, sekä FFD-heuristiikkaan. Testejä suoritettiin niin kutsutuissa homogeenisissä skenaarioissa, joissa virtuaalikoneiden suoritinteho- ja muistivaatimukset olivat suunnilleen yhtä korkeat, sekä haastavammissa skenaarioissa, joissa nämä vaatimukset olivat epätasaiset. Virtuaalikoneiden määrää vaihdeltiin välillä 1000–6000. Lopputuloksena oli, että tekijöiden esittämä algoritmi pystyi tuottamaan energiatehokkaampia ratkaisuja, varsinkin virtuaalikoneinstanssien määrän kasvaessa. [17] Alharbi, F et al. suorittivat testejä ympäristössä, jossa palvelimien kapasiteetit ja virtuaalikoneiden kapasiteettivaatimukset vaihtelivat. Myös he vertasivat omaa toteutustaan alkuperäiseen FFD-heuristiikkaan, sekä erilaisiin ACO-toteutuksiin. Heidän esittämänsä algoritmi saavutti merkittävät energiansäästöt, mutta hintana tälle oli se, että sen suoritusnopeus oli huomattavasti hitaampi kuin muilla ACO-algoritmeilla. [16] Myös Cao, C et al. vertasivat algoritmiaan FFD-heuristiikkaan, homogeenisessä skenaariossa, vaihtelevilla instanssien virtuaalikoneinstanssien määrällä. Myös heidän toteutuksensa saavutti paremman energiatehokkuuden, tosin sen suhteellinen paremmuus FFD-heuristiikkaan verrattuna ei juurikaan vaikuttanut kasvavan instanssien määrän noustessa. [18]

5. YHTEENVETO

Tämän työn tarkoituksena oli tarkastella, millaisia nykyaikaiset pilvipalvelut ovat, ja esittää yksi tapa, miten ne voidaan saada toimimaan ympäristöystävällisemmin. Aluksi lähdetään liikkeelle pilvipalvelujen määritelmästä, josta aletaan edetä teorian kautta lähemmäs ja lähemmäs varsinaista ydinasiaa: virtuaalikoneiden sijoittelua pilvipalveluissa.

Tämän kirjallisuuskatsauksen arvo jäi rajalliseksi, koska erilaisten algoritmitoteutusten syvällinen ja merkityksellinen vertailu olisi tehnyt tästä tutkielmasta liian suuren kokonaisuuden kandidaattitutkielmaksi. Kuitenkin voidaan todeta, että Ant Colony System –pohjaiset sijoittelumenetelmät ovat onnistuneet tuottamaan energiatehokkaampia ratkaisuja verrattuna yksinkertaisiin heuristiikkoihin. Näin ollen voidaan sanoa, että Ant Colony –optimoinnin, sekä muidenkin monimutkaisten optimointimenetelmien tutkimisella voi olla arvoa kestävän kehityksen edistämisessä.

Aiheeseen on useita näkökulmia, jotka tällaisessa työssä voisi huomioida. Mielenkiintoista olisi esimerkiksi ollut verrata Ant Colony –optimointia muihin luonnonilmiöiden inspiroimiin metaheuristiikkoihin. Se olisi mielestäni hyvä tutkimuksen aihe tulevaisuutta ajatellen. Mukaan olisi voinut ottaa myös kritiikin, jota metaheuristiikalle on kirjallisuudessa esitetty. Olisi ollut hyvä tehdä myös katsaus tulevaisuuteen: Tuleeko tekoäly tekemään sijoitteluprosessin ihmisten puolesta, tai tekevätkö vaihtoehtoiset teknologiat virtuaalikoneista täysin vanhentuneen teknologian pilvipalveluissa? Vertailua olisi voinut tehdä esimerkiksi konttiteknologiaan, bare-metal –pilveen, tai serverless-ratkaisuihin.

Aiheesta ei löytynyt kovin suurta määrää kirjallisuutta, mikä on osasyynä siihen, että tarkastelu jäi hieman suppeaksi. Toisaalta tämä tarkoittaa myös sitä, että tulevaisuudessa on reilusti tilaa uudelle tutkimukselle aiheeseen, Tähän olisi mielestäni myös aihetta, ottaen huomioon tutkimuksissa saavutetut lupaavat tulokset.

6. LÄHTEET

[1] *Cloud computing services by size class of enterprise*. (2024). Eurostat. haettu tammikuussa 19, 2025, osoitteesta https://ec.europa.eu/eurostat/databrowser/view/isoc_cicce_use/default/table?lang=en

[2] Mell, P. and Grance, T. The NIST Defnition of Cloud Computing. National Institute of Standards and Technology (2011).

[3] Höfer, C. N. & Karagiannis, G. (2011) Cloud computing services: taxonomy and comparison. Journal of internet services and applications. [Online] 2 (2), 81–94.

[4] Saini, H., Upadhyaya, A., & Khandelwal, M. K. (2019, October). Benefits of cloud computing for business enterprises: A review. In Proceedings of International Conference on Advancements in Computing & Management (ICACM).

[5] Xue, C. T. S., & Xin, F. T. W. (2016). Benefits and challenges of the adoption of cloud computing in business. *International Journal on Cloud Computing: Services and Architecture*, *6*(6), 01-15.

[6] Xing, Y., & Zhan, Y. (2012). Virtualization and cloud computing. In *Future Wireless Networks and Information Systems: Volume 1* (pp. 305-312). Springer Berlin Heidelberg.

[7] Portnoy, M. (2023) Virtualization Essentials. 3rd edition. Newark: John Wiley & Sons, Incorporated.

[8] Rashid, A. and Chaturvedi, A., 2019. Virtualization and its role in cloud computing environment. International Journal of Computer Sciences and Engineering, 7(4), pp.1131-1136.

[9] Masdari, M. et al. (2016) An overview of virtual machine placement schemes in cloud computing. Journal of network and computer applications. [Online] 66106–127.

[10] Silva Filho, M. C., Monteiro, C. C., Inácio, P. R., & Freire, M. M. (2018). Approaches for optimizing virtual machine placement and migration in cloud environments: A survey. Journal of Parallel and Distributed Computing, 111, 222-250.

[11] Saidi, K., & Bardou, D. (2023). Task scheduling and VM placement to resource allocation in Cloud computing: challenges and opportunities. *Cluster Computing*, *26*(5), 3069-3087.

[12] Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM computing surveys (CSUR)*, *35*(3), 268-308.

[13] Gogna, A., & Tayal, A. (2013). Metaheuristics: review and application. *Journal of Experimental & Theoretical Artificial Intelligence*, *25*(4), 503-526.

[14] Dorigo, M., Birattari, M., & Stutzle, T. (2006). Ant colony optimization. *IEEE computational intelligence magazine*, *1*(4), 28-39.

[15] Dorigo, M., & Gambardella, L. M. (1997). Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on evolutionary computation*, *1*(1), 53-66.

[16] Alharbi, F., Tian, Y. C., Tang, M., Zhang, W. Z., Peng, C., & Fei, M. (2019). An ant colony system for energy-efficient dynamic virtual machine placement in data centers. Expert Systems with Applications, 120, 228-238.

[17] Liu, X.-F. et al. (2018) An Energy Efficient Ant Colony System for Virtual Machine Placement in Cloud Computing. IEEE transactions on evolutionary computation. [Online] 22 (1), 113–128.

[18] Gao, C., Wang, H., Zhai, L., Gao, Y., & Yi, S. (2016). An energy-aware ant colony algorithm for network-aware virtual machine placement in cloud computing. In 2016 IEEE 22nd international conference on parallel and distributed systems (ICPADS) (pp. 669-676). IEEE.