TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

JAAKKO SADEHARJU

PILVIPOHJAINEN VIDEON KOODIMUUNNOKSEN HAJAUTTAMINEN

Diplomityö

Tarkastaja: Hannu-Matti Järvinen
Tarkastaja ja aihe hyväksytty
Tieto- ja sähkötekniikan tiedekunta-neuvoston kokouksessa 4. touko-kuuta 2016

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
SADEHARJU, Jaakko: Pilvipohjainen videon koodimuunnoksen hajauttaminen
Diplomityö, 46 sivua
Toukokuu 2016
Pääaine: Sulautetut järjestelmät
Tarkastaja: professori Hannu-Matti Järvinen
Avainsanat: Pilvipalvelu, videokoodimuunnos, laskennan hajauttaminen

Tässä työssä selvitetään miten pilvipalveluna tarjottavat alustat soveltuvat videon koodimuunnoksen toteuttavan sovelluksen hajauttamiseen. Tietokoneen laskennan muuttuviin laskentavaatimuksiin on kannattavaa sopeutua jollain tavalla. Yksi ratkaisu vaihtelevaan kuormitukseen sopeutumisessa on lisätä laskentakapasiteettia laskemalla jotkin osat rinnakkain erillisillä tietokoneilla. Tässä voidaan käyttää hyväksi internetissä tarjolla olevia julkisia pilvipalveluita, jotka tarjoavat joustavan tavan ottaa käyttöön uusi palvelin laskennan tueksi. Tämä mahdollistaa palvelinten käyttöönoton ilman suurta alkuinvestointia sekä niistä luopumisen tarpeen vähentyessä, koska pilvipalveluna tarjottavista palvelinalustoista maksetaan käytön mukaan. Pilvipalveluiden joustavuutta voidaan käyttää hyväksi erityisesti silloin, kun järjestelmän kuormitus on epätasainen ja vaikeasti ennustettava. Kun kuormitus on suuri, voidaan käynnistää uusia palvelimia ja kuormituksen ollessa vähäistä sulkea ylimääräiset tietokoneet kustannusten vähentämiseksi.

Laskennan hajauttamisen tutkimista varten toteutettiin mallijärjestelmä, joka toteuttaa videomuunnoksen hajautetun laskennan. Videon koodimuunnos tarkoittaa videon koon, laadun tai koodauksen muuntamista johonkin uuteen muotoon. Tämä muunnosprosessi voidaan hajauttaa erillisille laskentayksiköille jakamalla video lyhyisiin osiin. Näiden osien muunnos voidaan toteuttaa toisistaan riippumatta ja siksi ne voidaan laskea erillisillä tietokoneilla. Toteutettua järjestelmää hyväksi käyttäen mitattiin, miten rinnakkaisuuden määrä vaikuttaa koko järjestelmän tehokkuuteen. Videon koodimuunnosta suoritettaessa hajautuksesta on hyötyä, mikä tulee esille erityisesti silloin, kun laskentaa hajautetaan suhteellisen harvalle tietokonemäärälle. Mitä suuremmaksi hajautuksen määrä kasvaa, sitä pienemmäksi saavutettava laskentatehon lisäys jää.

Tärkeää hajauttamisen toteutuskelpoisuuden kannalta on myös siitä syntyvien kustannusten arviointi. Potentiaalinen hyöty on merkittävä, koska kuormituksen muuttuessa voidaan pitää yllä aina sopivaa määrää laskentatehoa. Kuormituksen monitorointi ja sen muutoksiin varautuminen asettaa kuitenkin haasteita laskentakapasiteetin optimoinnille. Jotta voidaan arvioida kulloisenkin laskentakapasiteetin riittävyyttä, on tiedettävä, miten suuri on kyseisen hetken kuormitus.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
SADEHARJU, JAAKKO: Distributing video transcoding using cloud based platforms
Master of Science Thesis, 46 pages
May 2016
Major: Embedded Systems
Examiner: Professor Hannu-Matti Järvinen
Keywords: cloud computing, video transcoding, distributed computing

In this thesis it is studied how suitable internet cloud services are in distributing video transcoding. Adapting to fluctuating computing load should be considered when designing a system. This can be realized by distributing computing into several separate computing units instead of using just one server. Internet cloud services can be used to adapt to changes in computing load demand. Cloud services enable flexible computing capacity by fast implementation of new servers for concurrent computing as well as shut down of excess servers when needed. This is achieved by hour based prizing which makes possible to implement new servers without high initial costs. The flexibility of the cloud services is especially useful in the situations where the load of the system is changing relatively much.

This thesis describes a test environment which implements a video transcoding system using cloud based computing instances. Transcoding is a process where a digital video is encoded to a different format i.e. different resolution bitrate or encoding. The encoding process may be distributed to separate computing instances by splitting the video into smaller sections. These sections may be encoded individually and that makes it possible to distribute the computing.

The performance of this testing environment was then measured using different amount of cloud computing instances. This measurement was analyzed to estimate how suitable the public cloud services are to the transcoding process and what is the optimal amount of concurrency. It was shown that the distribution to the cloud services are useful and it improves the performance of the transcoding system. The maximum boost to performance was achieved using relatively few cloud instances and the gain to the performance was getting smaller when the instance count was increased.

Another important aspect to the distribution of transcoding was to estimate the costs of using cloud services. The flexibility of the cloud services makes it a good option for transcoding because it may be adjusted to the need of computing power. The scalability of the system needs to be monitored to fully benefit of the cloud services. When this is taken care of the usage of cloud services brings savings when compared to traditional computing platforms.

ALKUSANAT

Tämä diplomityö on kirjoitettu osana Tampereen teknillisen yliopiston diplomi-insinöörin tutkintoa. Työn aiheeseen vaikutti pääasiassa oma mielenkiinto sekä työpaikallani käyty keskustelu videokoodauksen laskennan skaalaamisesta. Työ on toteutettu lähes kokonaan vapaa-ajalla, ja sen kirjoittaminen on vaatinut aikaa ja voimia. Työn valmistumisen edesauttamisesta haluan kiittää professori Hannu-Matti Järvistä tärkeistä neuvoista ja keskustelusta työn sisältöön ja aiheen rajaamiseen liittyen sekä työn tarkastamisesta ja korjausehdotuksista. Lisäksi haluan kiittää Jarkko Viikkiä hyödyllisistä keskusteluista ja motivoinnista työn edetessä.

Haluan kiittää myös vanhempiani koko opiskeluaikani kestäneestä tuesta. Erityisen suuren kiitoksen ansaitsee vaimoni Katri, joka on jaksanut kannustaa ja tukea, niin työn kirjoittamisessa kuin arjessakin.

Tampereella 9.5.2016

Jaakko Sadeharju

SISÄLLYS

TERMIT JA LYHENTEET

Avainkehys	Engl. key frame. Videon pakkauksessa ne pysäytyskuvat, joista tallennetaan kaikki kuvainformaatio. Avainkehysten välisistä pysäytyskuvista tallennetaan muutostieto suhteessa avainkehykseen.
Bittinopeus	Engl. bit rate. Videon informaation määrä ajan suhteen. Yksikkö b/s
H.264	Eräs videokoodausstandardi.
IaaS	Engl. Infrastructure as a Service. Palveluna tarjottava infrastuktuuri.
PaaS	Engl. Platform as a Service. Palveluna tarjottava alusta.
Pilvipalvelu	Internetissä palveluna tarjottava kokonaisuus. Ks. myös IaaS, PaaS ja SaaS.
SaaS	Engl. Software as a Service. Palveluna tarjottava ohjelmisto.
Transkoodaus	Prosessi, jossa videon tallennusmuoto muunnetaan toiseksi.
ZeroMQ	Sovellus, joka toteuttaa viestijonon käyttämällä erityyppisiä socketteja.

1 JOHDANTO

Tietotekniset sovellukset ovat kehittyneet siihen suuntaan, että yhä enenevässä määrin sovellusten toiminta tapahtuu internetissä sijaitsevilla palvelimilla, kun aiemmin ohjelmat käyttivät paikallista laskenta- ja tallennuskapasiteettia. Tätä kehitystä kuvaa hyvin esimerkiksi valokuvien tallennus- ja käsittelyprosessi. Vielä runsas vuosikymmen sitten valokuva tallentui fyysiselle filmirullalle, joka kehitettiin paperivalokuvaksi ja talletettiin kansioon. Nykypäivänä valokuvan voi ottaa esimerkiksi älypuhelimella ja se tallentuu automaattisesti pilvipalveluun, jossa sitä voidaan tarkastella, ja josta se voidaan noutaa satoihin eri sovelluksiin käyttäen hyväksi ohjelmistorajapintaa. Tällä tavalla kuvatiedosto voi päätyä lähes automaattisesti, alle minuutin kuluttua ottamisesta, esimerkiksi johonkin sovelluksen profiilikuvaksi, toiseen automaattisesti generoitavaa digitaalista valokuvaalbumia varten.

Suuri osa internetissä tapahtuvien ohjelmien käytöstä tapahtuu siis nykyään ulkoisilla palvelimilla. Käytännössä tänä päivänä käyttäjän laitteella tai selaimessa tapahtuu vain tiedon näyttäminen. Tiedon tallennus ja varsinainen prosessointi tapahtuu internetissä taustalla toimivalla palvelimella. Tällaista laskentaa kutsutaan pilvilaskennaksi, ja tätä hyväksi käyttävää sovellusta ja ympäristöä pilvipalveluksi. Perinteisellä ohjelmistomallilla sovellus ostettiin kaupasta, asennettiin omalle koneelle, käytettiin paikallisesti ja ohjelman tila ja tallennettava data oli tallessa käyttäjän tietokoneella. Internetiä saatettiin käyttää esimerkiksi tiedon hakemiseen, mutta ohjelma itsessään oli käytännössä kokonaan paikallisella levyllä. Pilvipalvelumallissa kaikkien sovellusta käyttävien henkilöiden omistama tieto sijaitsee yhteisillä palvelimilla, joista käyttäjän laitteella sijaitseva sovellus hakee ja tallentaa ohjelman tilan, henkilökohtaiset asetukset ja käytettävän datan.

Myös perinteisesti paljon dataa sisältäviä sovelluksia, kuten videovuokraamoa, musiikin kuuntelua, tietokonepelejä, kirjastoa sekä aiemmin mainittua valokuvakirjastoa, käytetään nykyään entistä enemmän pelkästään pilvipalveluihin tukeutuen. Näihin liittyy myös tämän työn aihe, joka käsittelee pilvipalvelussa tapahtuvaa laskennan hajauttamista. Koska internetissä samaa palvelua käyttää yhtäaikaisesti moni käyttäjä, nämä palvelut vaativat merkittävän määrän laskentatehoa. Mikään tietokone maailmassa ei ole tarpeeksi tehokas palvelemaan miljoonia käyttäjiä kerralla, eikä mikään tietoverkko ole tarpeeksi nopea, että yhdestä paikasta voitaisiin tarjota sisältöä miljoonille käyttäjille ympäri maailmaa. Tämä ongelma voidaan ratkaista hajauttamalla laskentaa ympäri maailmaa siten, että laskentatyö hajautetaan usealle eri palvelimelle ja vastaavasti useaan eri kaupunkiin ympäri maailmaa. Tällöin siis laskenta voidaan jakaa pienempiin osiin, ja vastaavasti siirtää lähemmäksi palvelun käyttäjää.

Pilvipalveluiden mahdollistama tiedon hajauttaminen on järkevää, koska laskentateho voidaan yhden yksikön sijasta hajauttaa monelle yksikölle, mutta kuitenkin pilvipalveluiden käyttö hyödyttää myös päinvastaisella tavalla. Koska kaikki käyttäjät käyttävät lopulta samaa sovellusta, ei sovellusta tarvitse jakaa joka asiakasta varten erillisille

koneille. Toisaalta pilvipalvelut siis mahdollistavat sovellusten hajauttamisen ympäri maailmaa ja toisaalta niiden keskittämisen yksittäisille palvelimille. Tällöin sovellusten käyttämiä palvelimia voidaan kuormittaa sopivasti, jolloin turhia palvelimia ei pidetä ajossa, mutta samalla laskentateho riittää suurillekin käyttäjämäärille.

1.1 Esimerkkitapauksena videokoodimuunnos

Laskennan hajauttamiseen ja pilvipalveluiden käyttöön liittyvät asiat ovat suurelta osin yhteisiä kaikelle internetissä tapahtuvalle laskennalle, mutta tässä työssä keskitytään tarkastelemaan sitä erityisesti videon koodimuunnoksen hajauttamisen osalta. Videon koodimuunnos tarkoittaa digitaalisen videon formaatin, toisin sanottuna koon, laadun ja pakkausalgoritmin muuntamista johonkin toiseen formaattiin. Tämä on tarpeellista esimerkiksi silloin, kun samaa tallennettua videota halutaan toistaa usealla erilaisella päätelaitteella. Esimerkiksi Full HD-kotiteatterijärjestelmä asettaa videon laadulle täysin erilaisia vaatimuksia kuin matkapuhelin. Molemmilta laitteilta voidaan haluta katsoa samaa videota, vaikka toisen näytön pinta-ala on moninkertainen, ja käytössä oleva verkkoyhteys voi olla huomattavasti nopeampi. Suuria näyttöjä ja nopeita verkkoyhteyksiä varten halutaan useimmiten hyvälaatuinen video, eikä tiedoston koosta tai laadusta tarvitse tinkiä merkittävästi. Sen sijaan matkapuhelimessa on yleensä pieni näyttö, ja se saattaa sijaita hitaassa tietoverkossa, jolloin halutaan nopeasti latautuva video, vaikka videon laatu ei olisi erityisen korkeatasoinen. Jotta tähän päästään on alkuperäinen video voitava palvella useassa eri formaatissa, ja siitä on siis laskettava useita erilaisia versioita, jotta voidaan mahdollisimman hyvin palvella asiakkaita erilaisissa tilanteissa.

1.2 Työn tavoitteet ja rakenne

Tässä työssä selvitetään, miten hyvin internetissä tarjottavat julkiset pilvipalvelut sopivat videokoodauksen hajautettuun laskentaan. Työssä tehdään esimerkkisovellus, jota hyväksi käyttäen voidaan tehdä mittauksia tämän kaltaisen järjestelmän suorituskyvystä sekä arvioita pilvipalveluiden käytöstä aiheutuvista kustannuksista. Tätä varten alun teoriaosuudessa käsitellään pilvipalveluiden toimintaa yleensä (luku 2), pilvessä tapahtuvaa laskentaa sekä laskennan vaakasuoraa skaalautumista ja hajauttamista erillisille laskentatietokoneille. Vaikka pilvipalveluiden käyttö ja laskennan hajauttaminen ei rajoitukaan tähän ongelmaan, käsitellään sitä tässä työssä erityisesti tältä näkökannalta. Tämän vuoksi teoriaosuudessa käsitellään myös digitaalisen videon toimintaa (luku 4).

Teorialukujen jälkeen, luvussa 5, kuvataan testijärjestelmän suunnittelua ja rakennetta. Esitellään käytetyt teknologiat ja järjestelmät, ja tuodaan esille myös vastaan tulleet mahdollisuudet ja rajoitukset. Luvussa 6 esitetään työssä tehdyt mittaukset, sekä niihin liittyvät tulokset. Luku 7 sisältää tehtyjen mittaustulosten analyysiä sekä sitä, mitä tehdyt mittaukset paljastivat testiympäristöstä ja käsiteltävästä ongelmasta yleensä.

Viimeisessä luvussa (luku 8), arvioidaan työn onnistumista kokonaisuudessaan, sitä miten työssä kuvatut teknologiat soveltuvat ongelman ratkaisuun, miten tutkimusta

kannattaisi jatkaa eteenpäin sekä mitkä osat työssä olisi kannattanut toteuttaa jollain toisella tavalla. Onnistuessaan työ antaa hyviä viitteitä siitä, miten hyvin pilvipalvelut soveltuvat videon hajauttamiseen ja miten kannattavaa tämän kaltainen hajauttaminen on.

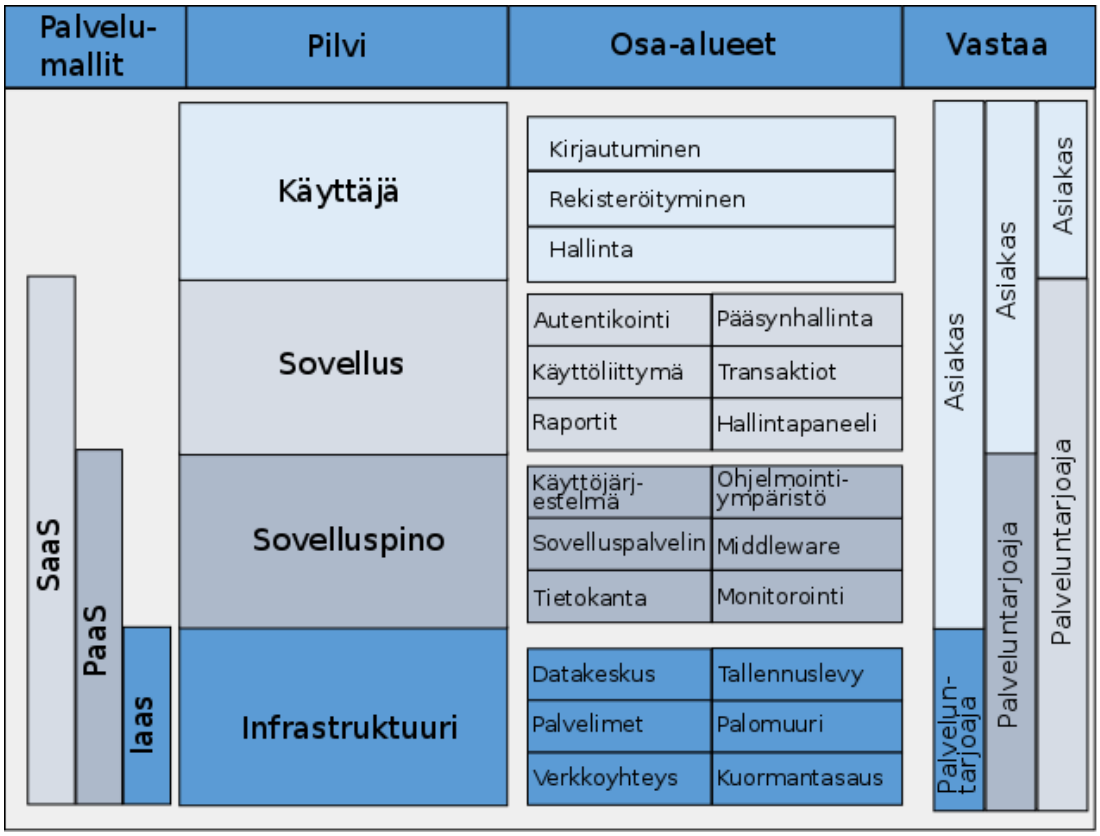
2 PILVILASKENTA

Pilvilaskenta on viime vuosikymmenen aikana merkittävästi kasvanut menetelmä, joka tarkoittaa laskennan hajauttamista Internetissä käyttäen hyväksi palveluna tarjottavia alustoja tai ohjelmistoja. Pilvilaskentaa voidaan pitää hajautetun laskennan erityistapauksena, ja suuri osa hajautetun laskennan teoriasta soveltuu myös pilvilaskentaan. Yhtä ainoaa määritelmää hajautetulle laskennalle ei ole, mutta järkevä oletus on, että laskentayksiköt ovat itsenäisiä, ja että laskentatehtävä jaetaan jollain tavalla näiden yksiköiden laskettaviksi. Seuraavissa luvuissa kuvataan pilvipalveluiden toimintaa yleensä sekä niiden ominaisuuksia. Pilvipalvelu voidaan jakaa kolmeen eri tason palveluun, joita tarjotaan erilaisina kokonaisuuksina: palveluna tarjottavat infrastruktuurit (engl. Infrastructure as a Service, IaaS), alustat (Platform as a Service, PaaS) ja ohjelmistot (Software as a Service, SaaS).

Edellä kuvattua jakoa selittää hyvin alla oleva kuva 2.1 [1]. Kuvan alaosassa olevat laatikot kuvaavat eri osa-alueita, jotka ovat kuuluvat palveluna tarjottavaan kokonaisuuteen, lukuun ottamatta vaaleimpia laatikoita, jotka kuvaavat sitä osaa kokonaisuudesta, joka jää asiakkaan vastuulle. Infrastruktuuri kuuluu osana palveluna tarjottavaan alustaan, ja vastaavasti alusta kuuluu osana palveluna tarjottavaan ohjelmistoon. SaaS siis sisältää sekä PaaSin että IaaSin, jolloin asiakkaan tarvitsee ymmärtää tarjottavasta kokonaisuudesta vähemmän. Käyttäjän vastuulle jää kaikissa tapauksissa ohjelman käyttö. Lähes kaikissa tapauksissa palveluna tarjottavaa sovellusta käytetään verkkoselaimella. Sovelluksen käyttö on aina tapauskohtaista, mutta yleensä siihen sisältyy järjestelmään rekisteröityminen, kirjautuminen ja hallinta.

Termiä ”pilvi” ehdotettiin ensimmäisen kerran käyttöön vuonna 2007, kuvaamaan Internetin käytössä tapahtunutta paradigman muutosta. Jo aiemmin oli ollut käytössä termi ”Software as a Service”. SaaSin lisäksi haluttiin laajempaa termiä, joka kattaisi myös laitteiston, levytilan ynnä muun sellaisen tarjoamisen internet-palveluna. Pilvi terminä kattaa siis sekä palveluna tarjottavat ohjelmistot että muunlaiset Internetissä tarjottavat resurssit. Näitä ovat siis palveluna tarjottavat alustat ja infrastruktuurit.

Pilvipalveluiden uskotaan tulevina vuosina yleistyvän huomattavasti. Tutkijat yleisesti uskovat, että etenkin SaaSin käyttö sovelluksissa tulee tulevaisuudessa kasvamaan huomattavasti, sillä se mahdollistaa joustavuuden ja lähes rajattoman laskentatehon [4]. Alaluvuissa 2.1, 2.2 ja 2.3 kuvataan edellä mainittuja osa-alueita ja niiden määritelmiä ja vastuualueita tarkemmin.



Kuva 2.1 *IaaS, PaaS ja SaaS sisällöt ja vastuualueet. Suomennettu lähteestä [1].*

2.1 Palveluna tarjottavat infrastruktuurit

Palveluna tarjottava infrastruktuuri tuo käyttäjän saataville sen, mikä perinteisesti on tarjottu datakeskuksessa. Siihen kuuluu käytännössä kaikki se laitteisto ja infrastruktuuri, joka vaaditaan käyttöjärjestelmän ylläpitämiseen. Tällaisia ovat itse palvelinlaitteisto (esimerkiksi suoritin, keskusmuisti ja kiintolevy), verkkoyhteys, palomuuri, sähköverkko, jäähdytetyt tilat ynnä muut sellaiset. Käytettäessä palveluna tarjottavaa infrastruktuuria palvelun käyttäjän vastuulle jää kaikki ylemmän tason toiminnallisuudet.

Palveluna tarjottavien infrastruktuurien käyttö on viime aikoina kasvanut huomattavasti, mistä kertoo muun muassa Amazon Web Servicesin merkittävä kasvu. Vaikka Amazonin palveluiden suosion kasvu on ollut suurempaa kuin IaaS-palveluntarjoajien keskimäärin, se kertoo vähintäänkin siitä, että infrastruktuurien tarjoaminen palveluna on kannattavaa liiketoimintaa [22]. Lisäksi se kertoo siitä, että näiden palveluille on kysyntää, ja että asiakasyrityksille on monesti kannattavampaa ostaa palvelut ulkopuolelta, jolloin saavutetaan huomattavasti omaa palvelininfrastruktuuria suurempi joustavuus. Tämä tarkoittaa mahdollisuutta ostaa palvelut vain hetkeksi tai mahdollisesti skaalata järjestelmää ja lisätä tehoa myös hetkellisesti kuormahuippujen ajaksi. Koska palvelun laskutus on yleensä tuntiperusteinen, voidaan järjestelmän kuormitusta tasata kellonaikojen mu-

kaan. Vastaavasti oma laitteisto ja ympäristö edellyttäisivät laitteiston hankkimista ja investointiin sitoutumista pitkäksi aikaa, usein vuosiksi eteenpäin. Tätä puolta on tarkemmin selitetty alaluvussa 2.6.

Palvelinympäristön hankintaan, asennukseen ja ylläpitoon liittyy merkittävä määrä huomioon otettavia asioita, joita voidaan vähentää hankkimalla se palveluna. Tällaisia asioita ovat muun muassa laitteiston hankinta, asennus ja ylläpito sekä näihin liittyvä asiantuntemus ja aika. Jos alustan tarve on jatkuva ja sen hankkimisesta aiheutuvat kustannukset voidaan jakaa koko laitteiston elinkaaren ajalle, tulee oman laitteiston hankkiminen todennäköisesti halvemmaksi kuin palveluna hankittavat alustat. Kuitenkin oman laitteiston asennuksen vaativa asiantuntemus saattaa tehdä IaaS-palvelusta tässäkin tapauksessa halvemman ratkaisun. Jos asennus, ylläpito tai muu konsultointi joudutaan ostamaan yrityksen ulkopuolelta, voi näistä aiheutua merkittäviä kustannuksia, mikä tässäkin tapauksessa puoltaa IaaSin käyttöä.

2.2 Palveluna tarjottavat alustat

Palveluna tarjottavat alustat (Platform as a Service, PaaS) tarkoittaa ohjelmistoalustan ulkoistamista, eli sen hankkimista palveluna. Kehitysalusta saadaan valmiina, jolloin voidaan keskittyä pelkästään sen päälle kehitettävään sovellukseen. Perinteisesti alustat on kehitetty itse hankkimalla laitteisto sekä tarvittava ohjelmisto sovelluksen kehittämistä ja tuotantokäyttöä varten. PaaS sisältää kaiken mitä IaaS, tai siis palvelinlaitteiston, verkon, konesalin ynnä muut, mutta tämän lisäksi myös käyttöjärjestelmän, tietokannan, ohjelmointiympäristön ja niin edelleen, mitkä ovat edellytyksenä ohjelmistokehitykselle.

Vaikka alustapalvelujen ulkoistamisella saavutetaan monia hyötyjä, on sillä kuitenkin myös huonoja puolia. Vaikka PaaSin merkittävä hyöty on, ettei alustaa tarvitse itse asentaa, tästä kuitenkin seuraa se, ettei alusta ole täysin vapaasti muokattavissa. Vaikka laitteisto- ja käyttöjärjestelmävaihtoehtoja on valittavana useita, ei kaikkia mahdollisia vaihtoehtoja voida tarjota, eikä asiakas voi täysin vapaasti kustomoida alustaa. Usein tarjottavat vaihtoehdot ovat riittäviä sovelluksen kehittämiselle, mutta joskus tämä kuitenkin saattaa estää tai haitata sitä merkittävästi [5].

Palveluna tarjottavan alustan käyttöönoton esteenä saattaa joskus olla myös tietoturva. Palveluntarjoajalla on pääsy alustaan ja sen myötä ohjelmistoon, joten on varauduttava siihen, että tällaisella alustalla toteutettaviin järjestelmiin on pääsy myös ulkopuolisilla henkilöillä. Tämä voi haitata ulkoisten alustojen käyttöä turvallisuuskriittisissä ja arkaluontoista tietoa sisältävissä sovelluksissa. [5]

2.3 Palveluna hankittavat ohjelmistot

Palveluna hankittavat ohjelmistot (Software as a Service, SaaS) on se, mistä perinteisesti on puhuttu pilvipalveluna. SaaS-ohjelmistoja ajettaessa on yleensä käytössä vain yksi tuotantoinstanssi, joka palvelee useaa asiakasta samalla kerralla. Jokainen asiakas käyttää

siis samaa järjestelmää, mutta näkee vain itselleen kuuluvan tiedon. Yleensä palvelun hinnoittelu tapahtuu käytön mukaan tilauksesta, eikä ohjelmistolisenssistä kuten ohjelmistot perinteisesti.

Kuten mainittiin edellisessä luvussa, SaaS sisältää kaiken sen, minkä PaaS ja IaaSkin. Tämä tarkoittaa, että palveluntarjoaja vastaa paitsi palvelinalustasta ja käyttöjärjestelmästä, myös muiden muassa sovelluksen ajamisesta ja ohjelmistopäivityksistä. Myös pääsynhallinta sekä tietoturvan hallintatyökalut tulevat osana palvelua. Käyttäjän vastuulle jää siis periaatteessa vain ohjelmiston käyttö, mikä voidaan nähdä kuvasta 2.1.

2.4 Rinnakkaistuminen

Laskentaa hajautettaessa on tärkeä selvittää, miten helposti laskentatehtävä voidaan jakaa laskettavaksi usealle eri prosessorille. Jotta hajauttamisesta olisi hyötyä, on laskentatehtävän oltava riittävän helposti jaettavissa pienempiin osaongelmiin, jotka voidaan laskea toisistaan riippumatta. Mitä suurempi on osien keskinäinen riippumattomuus, sitä suurempi hyöty saadaan laskennan hajauttamisesta.

Tällaisia ongelmia ovat esimerkiksi suurten matriisien laskentaoperaatiot, jotka voidaan jakaa alimatriiseihin, tai www-sivusto, jonka yksittäistä pyyntöä ei ole helppo jakaa osiin, mutta useat rinnakkaiset pyynnöt voidaan ohjata eri palvelimille. Myös alkulukujen etsintään käytetään satoja tuhansia tietokoneita, jotka selvittävät yksi luku kerrallaan onko kyseinen luku alkuluku. Teoriassa yhden luvun jaollisuuden selvittäminenkin voitaisiin vielä jakaa pienempiin osaongelmiin, jolloin yksi kone saisi testattavakseen vain osan kyseisen luvun mahdollisista jakajista. Teoriassa helposti rinnakkaistuva ongelma voidaan suorittaa n:llä laskentayksiköllä n:ssä osassa siitä ajasta mikä tehtävään kuluisi yhdeltä prosessorilta. [28, 29]

Toisessa ääripäässä ovat ongelmat, joita ei voida jakaa rinnakkain suoritettaviin osiin, jolloin laskutehtävän yhden vaiheen tulos on oltava selvillä ennen kuin seuraavaa vaihetta voidaan laskea. Jos ongelma on huonosti rinnakkaistuva, ei rinnakkaisella laskennalla saavuteta merkittävää hyötyä tai rinnakkaistaminen jopa hidastaisi laskentaa. Esimerkkejä tällaisesta ongelmasta algoritmit kuten esimerkiksi Newton-Raphsonin menetelmä, jossa matemaattisen funktion nollakohtaa pyritään estimoimaan iteroimalla.

Tässä työssä ei keskitytä ongelmien rinnakkaistamiseen, mutta on hyvä todeta, että videokoodauksen hajauttaminen, jota tässä työssä tarkastellaan, sijoittuu yllä kuvattujen ääripäiden väliin. Se on suhteellisen helposti rinnakkaistettavissa, mutta vaatii kuitenkin hieman ylimääräistä työtä hajautetun laskennan lisäksi. Video on jaettava osiin, ja valmiit osat liitettävä toisiinsa. Tämän takia videokoodaus sopii hyvin käytettäväksi tässä työssä.

2.5 Tiedonsiirto

Hajauttamisesta saatava hyöty riippuu myös tiedonsiirrosta eri yksiköiden välillä. Mitä vähemmän tietoa tarvitsee siirtää laskentayksiköiden välillä, sitä suurempi hyöty saadaan

hajauttamisesta. Vaikka itse laskenta olisikin helposti hajautettavissa usealle eri koneelle, voi hajauttamisen hyöty jäädä vähäiseksi, jos tarvittavan tiedonsiirron määrä on suuri.

2.6 Hyödyt

Pilvipalveluita käytettäessä saavutetaan monia hyötyjä verrattuna perinteiseen tapaan, jossa sovellus toimii palveluntuottajan omilla palvelimilla. Merkittäviä hyötyjä ovat muun muassa hinta, ketteryys ja ekologisuus. Näitä kolmea kuvataan seuraavissa alaluvuissa 2.6.1, 2.6.2 ja 2.6.3.

2.6.1 Hinta

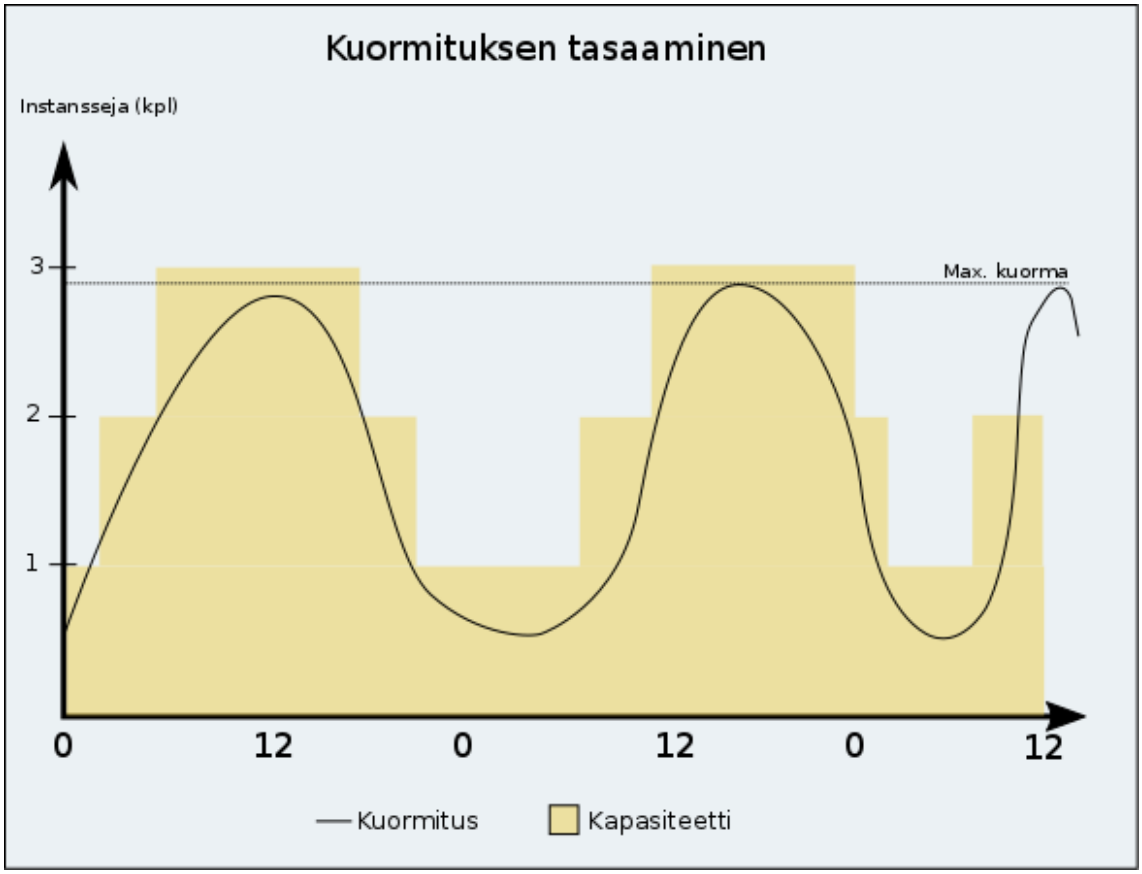
Pilvipalveluita käyttämällä voidaan usein saavuttaa säästöjä. Tämä ei ole itsestään selvää, mutta säästö syntyy periaatteessa samoista asioista kuin ulkoistamisessa yleensä. Kun yhden osa-alueen toimittaa taho, joka voi keskittyä vain tähän yhteen osa-alueeseen, sen asiantuntemus ja tehokkuus saadaan paremmaksi. Toisin sanottuna: ostettaessa palvelu ulkopuolelta ostetaan myös siihen liittyvä osaaminen. Kuten yritysmaailmassa yleisesti, pienen yrityksen ei yleensä kannata palkata omaa palkanlaskijaa tai siivoojaa, koska halutaan keskittyä vain oman yrityksen erityisosaamiseen. Myös ohjelmiston käytössä suurin hyöty saavutetaan, jos yrityksen kannalta epäolennainen tietotekninen osaaminen voidaan ulkoistaa.

Asiantuntemuksen lisäksi merkittävä hintaan vaikuttava tekijä on pilvipalveluiden ketteryys. Tämä tarkoittaa, että palvelusta voidaan maksaa käytön mukaan, eikä ylimääräisiä resursseja tarvitse pitää jatkuvasti saatavilla koko aikaa. Jos esimerkiksi ohjelmistoa tarvitaan vain yhteen aikaan vuodesta, voidaan ohjelmiston käyttöoikeus tilata vain joksikin kuukaudeksi, silloin kun sille on tarvetta. Muuna aikana voidaan toimia joko ilman järjestelmää, tai niillä ehdoilla ja ohjelmiston osilla, jotka ovat tarjolla ilmaiseksi. Vastaavasti ohjelmistosuunnittelija saattaa tarvita kehitykseen lisenssin ohjelmistoympäristöä varten tai laitealustan kehitystyötä varten. Näille ei välttämättä ole enää käyttöä kehitysvaiheen jälkeen. Seuraavassa alaluvussa käsitellään tarkemmin ketteryyden hyötyjä.

2.6.2 Ketteryys

Pilvipalveluiden tarjoamaa joustavuutta voidaan käyttää hyödyksi silloin, kun järjestelmän kuormitus eri aikoina vaihtelee suuresti. Usein pilvipalveluiden laskutus perustuu käyttöön, kuten luvussa 2.3. kerrottiin, joten yhden palvelimen pitäminen ajossa 24 tunnin ajan maksaisi saman verran kuin 24 palvelimen käyttäminen yhden tunnin ajan. Kun käyttöä on paljon, voidaan uusia palvelimia lisätä tasaamaan kuormitusta. Vastaavasti kun käyttö on vähäistä, voidaan ylimääräiset palvelimet poistaa käytöstä, jolloin ne eivät aiheuta kustannuksia. Mikäli järjestelmän kuormitusta pystytään luotettavasti mittaamaan,

ei ylimääräistä palvelinkapasiteettia tarvitse pitää ajossa. Näin saavutettu säästö on suuri etenkin järjestelmissä, joissa kuormitus vaihtelee runsaasti eri ajankohtina. Kuormituksen jakautumista on kuvattu kuvassa 2.2.



Kuva 2.2 *Palvelinkapasiteettia voidaan lisätä hetkellisesti kuormitushuippujen ajaksi.*

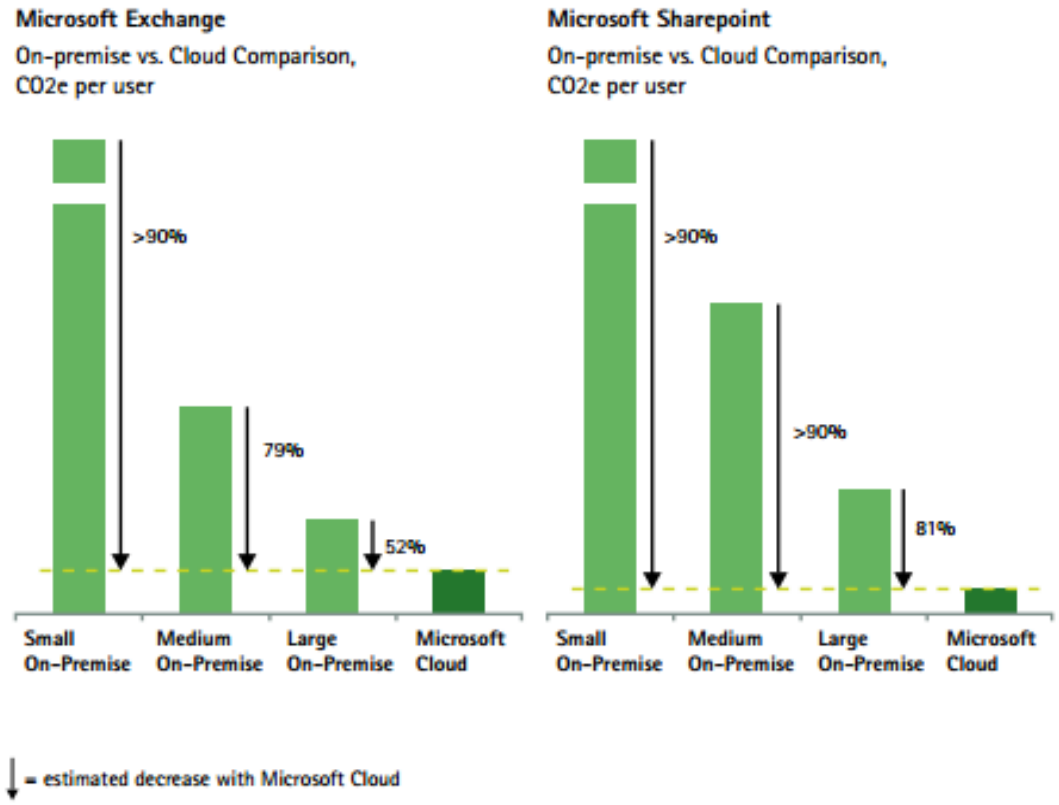
Kun järjestelmän kuormitus on korkeimmillaan, voidaan palvelinkapasiteettia lisätä hetkellisesti, eikä suurempaan kapasiteettiin tarvitse sitoutua pitkäksi aikaa. Vastaavasti ylimääräiset palvelimet voidaan poistaa käytöstä silloin, kun käyttö on vähäistä. Useimmissa järjestelmissä kuormahuiput osuvat päivälle, kun taas yöllä kuormitus on vähäisempää.

2.6.3 Ekologisuus

Vaikka palvelinkeskukset kuluttavat suuria määriä energiaa, sitä kuluu kuitenkin huomattavasti vähemmän kuin erillisissä, usean eri organisaation itselleen asentamissa palvelinsaleissa ja palvelimissa [2]. Pilvipalvelun energiatehokkuutta verrattuna perinteiseen palvelinympäristöön lisäävät ainakin seuraavat seikat: Dynaaminen varaaminen, moniasiakkaisuus, palvelimen käyttöaste ja palvelinsalien kehittynyt tehokkuus [23].

Dynaaminen varaaminen hyödyttää energiatehokkuutta, koska palvelinkapasiteettia voidaan varata tarpeen mukaan. Tällöin ei tarvitse turhaan varautua suureen kuormaan etukäteen, vaan kasvavaan tarpeeseen voidaan varautua vasta sen realisoituessa.

Moniasiakkaisuus hyödyttää palvelinkapasiteetin tehokasta käyttöä siten, että samalta palvelimelta voidaan palvella useaa asiakasta kerrallaan. Jokaiselle asiakkaalle ei siis tarvitse asentaa erillistä tietokonetta, vaan olemassa oleva palvelinlaitteisto voidaan jakaa halutulla tavalla usealle asiakkaalle. Tämä mahdollistaa osaltaan palvelimen tehokkaankäytön, koska koko käytettävissä olevaa kapasiteettia voidaan paremmin jakaa olemassa olevalle laitteistolle. Palvelinsalien suunnittelu voidaan tehdä tehokkaammaksi ja paremmaksi silloin, kun laitteisto on sijoitettu yhteen paikkaan. On tehokkaampaa tehdä jäähdytys vain yhteen tilaan usean erillisen tilan sijasta.



Kuva 2.3 *Microsoftin Exchange ja Sharepoint -sovellusten hiilidioksidipäästöjen vertailu perinteisen ja pilvipalvelumallin välillä. Muokattu lähteestä [23].*

Myös hiilidioksidipäästöt pienenevät huomattavasti, kun käytetään pilvipalvelimia perinteiseen palvelinkäyttöön verrattuna (on-premise). Microsoft ilmoittaa kolmen eri ohjelmistonsa hiilidioksidipäästöjen pienenevän merkittävästi siirryttäessä pilvipohjaiseen palvelumalliin (kuva 2.3). Esimerkiksi Sharepointin tapauksessa pieniin ja keskisuuriin yksiköihin nähden päästöt ovat vähentyneet yli 90 prosenttia ja suuriinkin yksiköihin verrattuna päästöjen arvellaan vähentyneen 81 prosenttia. [23]

3 SKAALAUTUVUUS

Verkkopalvelun käytön kasvaessa myös palvelinympäristöltä vaadittavat resurssit kasvavat. Vastaavasti myös käytön vähentymisestä johtuvaan resurssien vähentymiseen on tärkeää reagoida. Tähän resurssitarpeen muutokseen on vastattava, joko ohjelmiston arkkitehtuuria tehostamalla tai lisäämällä sovelluksen käytössä olevien resurssien määrää. Sovelluksen arkkitehtuuria muuttamalla saavutetaan merkittävin hyöty silloin, kun olemassa oleva toteutus on huonosti suunniteltu. Jo ennestään järkevästi toimivan sovelluksen tehostaminen muuttamalla arkkitehtuuria on usein liian työlästä saavutettuun hyötyyn verrattuna. Tämänlaisessa tapauksessa on lisättävä palvelun käytössä olevien resurssien määrää.

Sovelluksen käytössä olevien resurssien sopeuttamista käytännön tarpeen mukaan sanotaan skaalautuvuudeksi. Järjestelmän skaalautuvuus voidaan toteuttaa joko pystysuoraan, olemassa olevien yksiköiden tehoa tai kapasiteettia lisäämällä, tai vaakasuoraan, lisäämällä järjestelmään uusia erillisiä yksiköitä.

3.1 Pystysuora skaalautuminen

Kun verkkopalvelun käyttö kasvaa ja tarvitaan lisää laskentatehoa, muistia sekä tallennustilaa, voidaan palvelimen kapasiteettia vastaavasti lisätä. Voidaan vaihtaa tehokkaampi suoritin, lisätä muistia, tai levytilaa. Tämän kaltaista sopeutumista tehonlisäykseen kutustaan pystysuoraksi skaalautuvuudeksi (engl. vertical scaling).

Palvelimen suoritin voidaan vaihtaa tehokkaampaan, jotta pystytään vastaamaan lisääntyneeseen kapasiteetin tarpeeseen. Merkittävin resurssi verkkopalveluiden sulavan toiminnan kannalta on riittävä keskusmuistin määrä. Vaikka järjestelmän ajoympäristö saataisiin vastaamaan tarvittavaa tehonlisäystä, järjestelmän käyttö on harvoin tasaista kaikkina päivinä viikossa tai ympäri vuorokauden. Tämänlaisiin muutoksiin kapasiteetin tarpeessa on käytännössä mahdoton sopeutua pystysuoraan skaalattaessa.

3.2 Vaakasuora skaalautuminen

Kasvaneeseen kapasiteetin tarpeeseen voidaan vastata myös lisäämällä erillisten palvelimien määrää. Tällöin ei lisätä yksittäisen palvelimen suorituskykyä, vaan lisätään rinnalle toinen yksikkö.

Kun järjestelmää skaalataan tällä tavalla, myös sovelluksen arkkitehtuurin on oltava tämän kaltaiseen ympäristöön sopiva. Tämä sopivuus riippuu käytännössä laskettavan ongelman luonteesta. Jos laskentatehtävää ei voida hajauttaa eri säikeisiin, ei vaakasuorasta skaalautumisesta ole hyötyä. Vastaavasti, jos ongelma on helppo jakaa osaongelmiin, jolloin se on helppo jakaa osiin erillisille säikeille, on vaakasuora skaalaaminen

järkevää. Monet algoritmit ovat näiden kahden edellä kuvatun laskentaongelman välimuotoja, joista voidaan jakaa osiin jokin osa, kun taas jokin toinen osa joudutaan suorittamaan tietyssä järjestyksessä.

3.3 Kuormituksen mittaaminen

Käytettävän palvelinympäristön on kyettävä tarjoamaan sovellukselle riittävästi suorituskykyä kulloisenkin käytön mukaan. Tähän sisältyy useita haasteita, joista yksi on käyttäjämäärän ja sovelluksen kuormituksen ennustaminen. Kuormituksen ennustamisen luonne riippuu paljon käyttäjämäärästä.

Silloin, kun käyttäjiä paljon, järjestelmän kuormitusta tiettynä ajankohtana on helpompi ennustaa. Tällöin voidaan mitata järjestelmän keskimääräinen kuormitus kulloisenakin ajankohtana sekä pitää riittävästi resursseja käynnissä, jotta järjestelmä toimisi riittävän sujuvasti. Kun käyttäjiä on paljon, yksittäisen käyttäjän vaikutus kokonaisuuteen on pieni, ja järjestelmän käytön ennustettavuus paranee. Esimerkiksi, jos järjestelmällä on tiettynä ajanhetkenä keskimäärin sata käyttäjää, joista kaikki käyttävät järjestelmää keskenään saman suuruisella kuormalla, tällöin yksi uusi käyttäjä lisää palvelimen kuormaa yhden prosentin. Vastaavasti jos käyttäjiä on tuhat, uusi käyttäjä lisää kuormitusta 0,1 prosenttia. Tämänlaisessa tapauksessa järjestelmän oikean toiminnan kannalta kriittistä onkin riittävän tarkka ennuste senhetkisestä käytöstä. Mitä enemmän käyttäjiä on samanaikaisesti, sitä pienempi on yksittäisen käyttäjän vaikutus kokonaisuuteen.

Silloin kun käyttäjiä on vähän, järjestelmän kuormituksen riittävän tarkasta ennustamisesta tulee vaikeampaa tai jopa mahdotonta. Ääritapauksessa järjestelmässä on vain yksi käyttäjä, joka käyttää järjestelmää täysin satunnaisesti siten, että kuormitus on joko 0 tai 100 prosenttia. Tässä tapauksessa järjestelmä on joko täysin kuormitettu tai täysin ilman työtä. Tällöin kuormitusta ei voida luotettavasti ennustaa, ellei voida ennustaa yksittäisen käyttäjän toimintaa. Tämä voisi tapahtua esimerkiksi järjestelmän muiden osien tai jonkin toisen järjestelmän tarjoamien tietojen perusteella. Tämä on kuitenkin monesti mahdotonta ja järjestelmän on varauduttava yhtäkkiseen kuormituksen kasvuun.

3.4 Kuormantasaus

Usean palvelimen rinnakkaisissa järjestelmissä on tehtävät työt jaettava jotenkin olemassa olevien palvelinten välillä. Suurikin määrä rinnakkaisia tietokoneita voi olla täysin turhia, jos vain yhtä niistä käytetään työn suorittamiseen. Jotta alaluvussa 3.2. käsitelty, vaakasuora skaalautuminen voisi käytännössä toimia, on kuorma jaettava järkevästi vapaana olevien palvelinten välillä. Kuormantasausta (engl. load balancing) käytetään, jotta vapaana olevat resurssit saataisiin mahdollisimman hyvin ja tasaisesti käyttöön. Verkkopalvelun tapauksessa pyynnöt pyritään ohjaamaan aina vähiten kuormitetulle palvelimelle.

3.4.1 Algoritmit

Kuormantasaukseen on kehitetty useita algoritmeja, kuten kiertovuorottelu, yhteyksien määrään tai vapaisiin resursseihin perustuva vuorottelu, ennustava, satunnainen sekä painotetusti satunnainen algoritmi. Nämä ovat eritelty taulukossa 3.1.

Taulukko 3.1 *Kuormantasausalgoritmit*

Algoritmi	Kuvaus
Kiertovuorottelu	Uudet yhteydet jaetaan palvelimille pysyvässä järjestyksessä.
Yhteyksien määrä	Uusi yhteys ohjataan palvelimelle, jolla on vähiten avoimia yhteyksiä.
Ennustava	Perustuu yleensä kiertovuorotteluun tai yhteyksien määrää hyödyntävään algoritmiin, johon on lisätty sovelluskohtainen osa lisäämään vakautta.
Vapaat resurssit	Kuormantasaus tehdään vapaiden resurssien perusteella.
Satunnaisuus	Palvelin valitaan satunnaisesti.
Painotettu satunnaisuus	Sama kuin satunnainen, mutta palvelimille on annettu painokerroin resurssien mukaan.

Yksinkertaisin taulukon algoritmeista on kiertovuorottelu (engl. round robin), jossa yhteydet jaetaan palvelimille kiinteässä järjestyksessä. Tämä jakaa kaikille saman määrän pyyntöjä, mutta ei takaa, että pyynnöt kuormittaisivat palvelimia samassa suhteessa. Yhteyksien määrään perustuva kuormantasaus perustuu avoinna olevien yhteyksien määrään, jolloin uusi pyyntö ohjataan palvelimelle, jolla on vähiten avoimena olevia yhteyksiä. Tämä jakaa todellista kuormaa todennäköisesti tasaisemmin, kuin kiertovuorottelu, mutta ei edelleenkään takaa sitä, että pyyntö ohjattaisiin vähiten kuormitetulle palvelimelle. Niin kutsuttu ennustava kuormantasausalgoritmi pyrkii ottamaan huomioon sovelluksen toimintaa. Tämän algoritmin perustana on yleensä jompikumpi edellisistä algoritmeista, mutta pyrkii ennustamaan pyyntöjen aiheuttamaa kuormitusta ottamalla huomioon sovelluksen ominaisuuksia. Tällöin päästään hieman edellisiä parempaan kuormantasaukseen.

Edellisistä hieman poikkeava tapa toteuttaa kuormantasaus on mitata palvelimien vapaat resurssit ja tehdä kuormantasaus sen perusteella. Tämä on kuitenkin monimutkaisempi toteuttaa, kuin edellä kuvatut algoritmit, koska tämä vaatii monitorointitietojen välittämistä kuormantasauspalvelimelle. Tällä tavalla voidaan kuitenkin ohjata pyyntö palvelimelle, jolla on vähiten kuormaa, ja näin toteuttaa kuormantasaus tehokkaasti.

Joskus on käytössä myös satunnaisalgoritmi, jota käytettäessä pyynnöt ohjataan satunnaisesti jollekin palvelimelle. Tämä ei eroa oleellisesti kiertovuorottelusta, kun pyyntöjen määrä on suuri. Tästä hieman muunneltua algoritmia kutsutaan painotetuksi satunnaisuudeksi, joka toimii kuten satunnaisuusalgoritmi, mutta painottaa palvelimia vapaana olevien resurssien perusteella. Tämä on siis resurssi- ja satunnaisuusperusteisen algoritmin välimuoto.

3.4.2 Laskentayksikön lisääminen

Kun olemassa järjestelmän suorituskyky ei riitä, joudutaan lisäämään uusia laskentayksiköitä järjestelmään. Tällöin joudutaan kuitenkin miettimään, millä tavalla uusi palvelin otetaan käyttöön ja miten olemassa oleva kuorma jaetaan vanhojen ja uusien palvelinten kesken. Periaatteessa toimiva ratkaisu kuormantasaukseen on ohjata pyynnöt aina palvelimelle, jolla on kyseisellä hetkellä vähiten kuormaa. Palvelimia lisättäessä tämä ei kuitenkaan ole välttämättä aina yksinkertaista.

Monesti verkkopalvelun tapauksessa palvelimelle lähetetyt pyynnöt saattavat kestää joitain millisekunteja, ja niitä voi tulla tuhansia yhden sekunnin aikana. Tämä aiheuttaa haasteen uuden palvelimen lisäämiselle ja sille miten uudet pyynnöt tulisi ohjata palvelimille. Palvelinten kuormitusta mitattaessa joudutaan aina mittaamaan keskiarvoa jollain aikavälillä. Esimerkiksi Linux-palvelimen kuormasta kertovalla keskiarvokuormatiedolla (engl. load average) ilmoitetaan se, kuinka moni prosessi joutuu keskimäärin odottamaan ajoon pääsyä tietyllä aikavälillä. Nämä ajat ovat yleensä yksi, viisi ja viisitoista minuuttia. Mikäli kuormitusta seurataan tällä mittarilla ja käytössä olevien, kuormitettujen palvelinten sijaan pyynnöt ohjataan uudelle vähiten kuormitetulle palvelimelle, kuluu huomattavan pitkä aika, kun kaikki uudet pyynnöt ohjataan vain tälle yhdelle palvelimelle. Pelkästään kuormitukseen perustuva kuormantasaus onkin harvoin optimaalinen. [6]

Pyynnöt voidaan myös tasata palvelinten välillä avointen yhteyksien perusteella. Tällöin uusi pyyntö ohjataan sille palvelimelle, jolla on vähiten avoimia yhteyksiä. Tämä tapa ei kuitenkaan ota huomioon palvelinten todellisia resursseja tai kuormaa. Yhdelle palvelimelle voi tulla huomattavasti vähemmän pyyntöjä kuin toiselle, mutta se kuormittaa silti enemmän järjestelmää. Yhden hitaan pyynnön aikana saatetaan pystyä palvelemaan satoja, esimerkiksi staattisia pyyntöjä, jotka eivät vaadi palvelimelta laskentatyötä, vaan ne voidaan tarjota suoraan levyltä. Myös erot palvelinten absoluuttisessa suorituskyvyssä jää avointen yhteyksien määrään perustuvassa kuormantasauksessa huomioimatta.

4 DIGITAALISEN VIDEON OMINAISUUDET JA TOISTAMINEN

Nykyiset kuluttajakäytössä olevat videontoistolaitteet toistavat Full HD -resoluutioista videota. Pakkaamattomana tämän tasoisen videon yksi pysäytyskuva tarvitsee 6,22 MB levytilaa ja minuutti videota noin 9 GB. Tämän kaltaisen videon bittinopeus on 1,19 Gbps. Jo runsas levytilan käyttö riittäisi syyksi pakata videotiedostot, mutta vielä tärkeämpää tämä on tiedonsiirron kannalta. Sellaista tiedonsiirtokanavaa, jota pitkin voisi siirtää Full HD -tasoista videota pitkiä matkoja pakkaamattomana, ei ole olemassa. Tämä ongelma korostuu käytettäessä langattomia päätelaitteita, kuten matkapuhelimet ja kannettavat tietokoneet. [8]

4.1 Digitaalisen videon prosessointi ja pakkaus

Videota pakattaessa siitä pyritään poistamaan redundantti informaatio ilman, että tästä aiheutuu haittaa videon käytölle. Se kuinka paljon videota voidaan pakata, riippuu käyttötarkoituksesta. Mobiilitoistoa varten videon ei tarvitse olla yhtä virheetön kuin vaikkapa lääketieteellistä diagnoosia tehtäessä videon perusteella. Videota pakattaessa pyritään saamaan mahdollisimman hyvä pakkaussuhde niin, että videon laatu soveltuu käyttötarkoitukseensa. Pakkaussuhde voidaan ilmaista alkuperäisen videon ja pakatun videon bittinopeuksien suhteena. [7]

$$Pakkaussuhde = \frac{Alkuperäisen\ videon\ bittinopeus}{Pakatun\ videon\ bittinopeus}$$

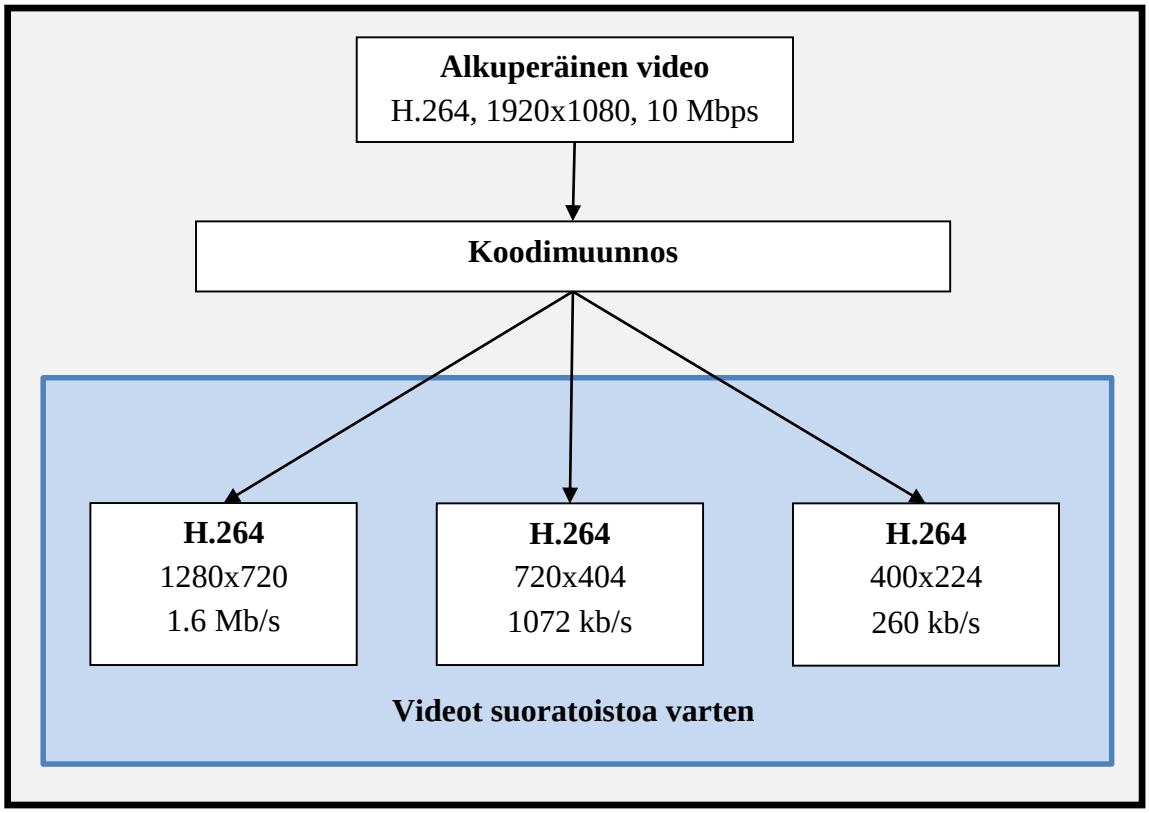
Käytettäessä häviötöntä pakkausta videon kokoa pyritään pienentämään siten, että pakkauksessa ei menetetä informaatiota. Pakattu video sisältää tällöin saman informaation kuin alkuperäinenkin video. Tästä on hyötyä varsinkin, jos pakattua videota joudutaan käsittelemään myöhemmin uudelleen.

Käyttämällä häviötöntä pakkausta voidaan alkuperäinen koko saada pienenemään noin puoleen alkuperäisestä. Häviöllistä pakkausta käytettäessä videon koko saadaan vielä huomattavasti pienemmäksi. Häviöllisessä pakkauksessa voidaan saavuttaa jopa noin 1/10 – 1/1000 pakkaussuhde. [9]

4.2 Koodimuunnos

Tässä työssä koodimuunnoksesta kirjoitettaessa tarkoitetaan ääni- tai videotiedoston koodauksen (engl. transcoding) tai bittinopeuden muuttamista (engl. transrating) toiseksi. Joskus nämä kaksi on mielekästä erottaa käsitteellisesti toisistaan [16], mutta tämän työn kannalta sitä ei ole tarpeen tehdä. Se muunnetaanko pelkästään bittinopeutta vai myös

koodausta vaikuttaa muunnoksen nopeuteen, mutta ei ratkaisevasti tässä diplomityössä suunniteltuun järjestelmään. Tässä työssä myös koon muuttaminen (engl. transsizing) kuuluu kiinteänä osana koodinmuunnokseen. Kuvassa 4.1 on esitetty H.264-standardilla toteutettu koodinmuunnosprosessi, jossa video muunnetaan kolmeen eri formaattiin. H.264 on laajasti käytössä oleva videokoodausstandardi. Lopputuloksena saatuja videoita voitaisiin käyttää suoratoistamiseen.



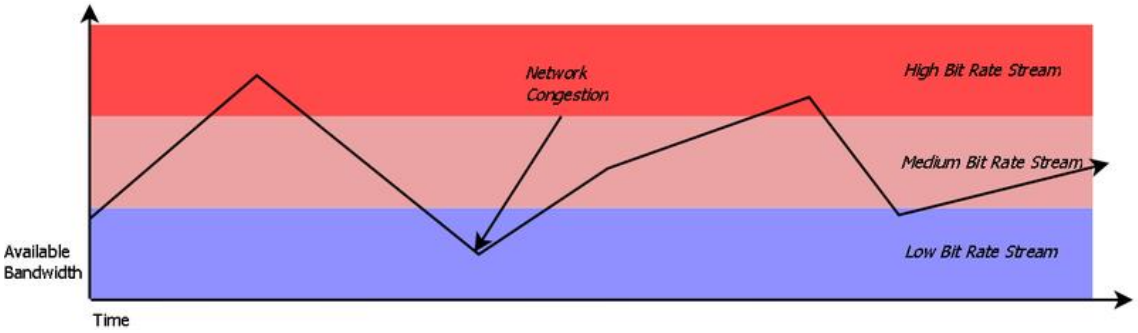
Kuva 4.1 *Videotiedoston koodimuunnos suoratoistoa varten.*

Internetissä jaettaville tiedostoille tehdään koodimuunnosta yleensä siitä syystä, että tiedosto halutaan muuntaa formaattiin, joka voidaan paremmin välittää tietoverkon yli. Muunnoksessa tehtävästä pakkauksesta aiheutuu kuitenkin aina häviötä. Pakkaukseen liittyvät artefaktit kumuloituvat, jos koodimuunnos tehdään samalle materiaalille useita kertoja peräkkäin. Tästä syystä videon ylimääräisiä koodimuunnoskertoja tulisikin välttää. Sama koskee kaikkea häviöllistä tiedon pakkaamista.

4.3 Adaptiivinen bittinopeus

Kun videota toistetaan verkon yli, joudutaan tekemään kompromissi videon laadun ja verkon nopeuden välillä. Jos video on hyvälaatuinen, se vaatii nopean verkon, eikä ole toistettavissa esimerkiksi matkapuhelinverkossa, jossa on huono kuuluvuus. Jos taas video on voimakkaasti pakattu ja pieniresoluutioinen, sen katseleminen isolta ruudulta nopealla internetyhteydellä ei tuota parasta saavutettavissa olevaa katselukokemusta.

Sama video voidaan tarjota käyttäjälle esimerkiksi matala- ja korkearesoluutioisena, jolloin käyttäjä saa valita videon oman laitteiston ja verkkoyhteyden suorituskyvyn mukaan. Tämä tapa on usealla sivustolla käytössä, mutta siinä on ongelmana se, että käyttäjä ei välttämättä etukäteen tiedä, miten hyvälaatuista videota hän voisi ladata. Tämä ongelma voidaan ratkaista toistamalla video adaptiivisella bittinopeudella (engl. adaptive bitrate streaming). Käytettäessä adaptiivista bittinopeutta käyttäjän Internet-yhteyden kaistanleveys mitataan videota toistettaessa. Videon laatua voidaan vaihtaa verkon todellisen suorituskyvyn mukaan kesken toistamisen. Tällöin videon valinta ei jää käyttäjän vastuulle ja voidaan sopeutua verkon muuttuvaan suorituskykyyn mikä on tavallista käytettäessä matkapuhelinverkkoa. Kuvassa 4.2 on esitetty kolmen eri videolaadun välittäminen katsojalle, jonka verkon suorituskyky vaihtelee katselun aikana.



Kuva 4.2 *Videon katselu adaptiivisella bittinopeudella [26].*

4.3.1 Vaatimukset videoformaatille

Jotta videon toisto olisi sujuvaa kaikilla päätelaitteilla, on varauduttava lähettämään katselijalle video useassa mahdollisessa formaatissa. Kun videota katsotaan kiinteän Internet-yhteyden yli esimerkiksi tietokoneella tai televisiolla, joka toistaa Full HD –tasoista kuvaa, on videolle asetetut vaatimukset täysin erilaiset kuin kannettavalle laitteilla katseltaessa. Esimerkiksi laajasti käytetyn Vimeo-sivuston pakkaussuosituksen mukaan video tulisi pakata joihinkin neljästä taulukossa 4.1 listatuista formaateista [11]. Nämä pakkausformaatit on tarkoitettu palveluun lähetettävälle videolle. Nämä eivät välttämättä ole lopullisia käyttäjälle lähetettäviä formaatteja, vaan käyttäjälle lähetetään koodimuunnettu versio päätelaitteen ja yhteyden mukaan. Videoiden bittinopeus on suhteellisen suuri, koska näitä ei ole tarkoitus välittää käyttäjälle sellaisenaan, vaan mahdollistaa hyvä laatu, kun video koodataan uudelleen eri bittinopeudelle. Tästä syystä palveluun ladattaessa suositellaan käyttämään hyvälaatuista videota, joka ei silminnähden kärsi uudelleenkoodauksesta.

Taulukko 4.1 *Vimeon käyttämät koodausformaatit*

Formaatti	Resoluutio	Bittinopeus
Standard Definition (SD) 4:3 kuvasuhde	640 x 480 px	2,000 – 5,000 kb/s
Standard Definition (SD) 16:9 kuvasuhde	640 x 360 px	2,000 – 5,000 kb/s
720p HD Video 16:9 kuvasuhde	1280 x 720 px	5,000 – 10,000 kb/s
1080p HD Video (Plus/PRO) 16:9 kuvasuhde	1920 x 1080 px	10,000 – 20,000 kb/s

Brightcave-yrityksen esittämät videoiden suoratoistoon käytettävät suositukset on listattu taulukossa 4.2. Tämän suosituksen mukaan koodattuja videoita ei ole tarkoitettu enää uudelleen prosessoitavaksi, vaan nämä ovat suosituksia käyttäjille lähetettävästä videosta. Suhteellisen hitaalla verkkoyhteydellä siirtämiseen optimoitua videota ei ole enää suotavaa muuntaa uuteen formaattiin, koska useaan kertaan koodattaessa videoon tulevat virheet kumuloituvat, kuten aiemmin kerrottiin.

Taulukko 4.2 *Brightcaven suositukset videon koodaukseen*

Resoluutio 4:3	Resoluutio 16:9	Bittinopeus
1280x960	1280x720	1.6 Mb/s
720x540	720x404	1072 kb/s
640x480	640x360	704 kb/s
480x360	480x268	436 kb/s
400x300	400x224	260 kb/s
400x300	400x224	110 kb/s

Johtuen koodauksen tyypistä (Mod4) Brightcaven suosituksessa videoiden resoluutioiden kaikki mitat ovat neljän monikertoja [10]. Tämä perustuu makroblokin 4x4 kokoon, joka on määritelty H.264/AVC -standardissa. AVC-standardissa määritellään useita mahdollisia makroblokin kokoja, joita ovat esimerkiksi 8x8, 4x4, 4x8 ja 8x4, mutta nämä kaikki perustuvat kuitenkin kokoon 4x4 [14]. Tästä seuraa, että laajakuvaresoluutioiset videot eivät ole täsmälleen 16:9 suhteessa. Laajakuvaresoluutioksi onkin valittu lähimpänä tätä suhdetta oleva neljän monikerta.

Katseltavan videon laatua voidaan vaihtaa vain avainkehysten kohdalla, koska avainkehysten välillä olevia kuvankaappauksia ei voida laskea ilman aikaisempaa avainkehystä. Lisäksi adaptiivista bittinopeutta käytettäessä kaikissa saman videon eri koodauksissa avainkehysten on oltava samoissa kohdissa. Mikäli avainkehykset on tallennettu eri kohtiin tai vaihdetaan muualla kuin avainkehyksen kohdalla, ei pysäytyskuvia ennen seuraavaa avainkehystä voida näyttää oikein. Tästä seuraa käyttäjälle näkyvää häiriötä.

4.3.2 Bittinopeuden vaihtelu

Video voidaan koodata joko siten, että bittinopeus pysyy vakiona, tai siten, että bittinopeus vaihtelee ajan myötä. Kun käytettävä bittinopeus on vakio, käytetään termiä vakiobittinopeus (engl. constant bitrate, CBR), ja kun bittinopeus vaihtelee ajan myötä riippuen videon sisällössä tapahtuvista muutoksista, käytetään termiä vaihtuva bittinopeus (engl. variable bitrate, VBR). Kun käytetään vakiobittinopeutta, video pakataan siten, että videon jokaisena hetkenä sen bittinopeus pysyy vakiona. Tämä ominaisuus vaaditaan, kun käytetään adaptiivista bittinopeutta.

4.3.3 Yksi- ja monikertainen koodaus

Videotiedosto voidaan koodata kokonaan yhdellä kertaa (engl. single-pass) tai useassa vaiheessa (engl. multi-pass). Useaan kertaan koodaamalla videon laatu saadaan paremmaksi, mutta tämä vaatii koko videon prosessoimisen useaan kertaan. Tällöin ensimmäisellä kerralla videon sisältö analysoidaan, jotta jälkimmäisillä kerroilla voidaan ottaa huomioon videon sisältö ja pakata sisältö eri tehokkuudella riippuen itse sisällöstä edellisen alaluvun kuvailemalla tavalla. Moninkertainen koodaus on kuitenkin yksinkertaista koodausta hitaampaa. [17]

Jos video koodataan vakiobittinopeudella, menetetään useaan kertaan prosessoinnista saavutettava hyöty, koska tällöin bittinopeutta ei voida muuttaa kesken koodausprosessin. Tämä käytännössä rajaa pois mahdollisuuden käyttää vaihtuvaa bittinopeutta livelähetyksissä. [17]

4.3.4 Suoratoisto ja progressiivinen lataus

Suoratoistoa käytettäessä video ladataan verkosta katselun aikana, joten tiedostoa pitää voida ladata nopeammin kuin videon katselu etenee. Mikäli videon bittinopeus on keskimäärin suurempi kuin verkon tiedonsiirtokapasiteetti, on odotettavissa, että käyttäjä joutuu jossain vaiheessa katselua odottamaan videon latautumista. Tähän vaikuttaa kuitenkin oleellisesti videon pituus ja tiedonsiirtopuskurin koko. Mikäli video on lyhyt, on mahdollista, että katselun alussa ladattava puskuri ei ehdi loppua, vaikka verkon suorituskyky olisi pienempi kuin videon bittinopeus. Mitä suurempi videon pituus on, sitä epätodennäköisempää on, että voitaisiin luottaa suureen latauspuskuriin.

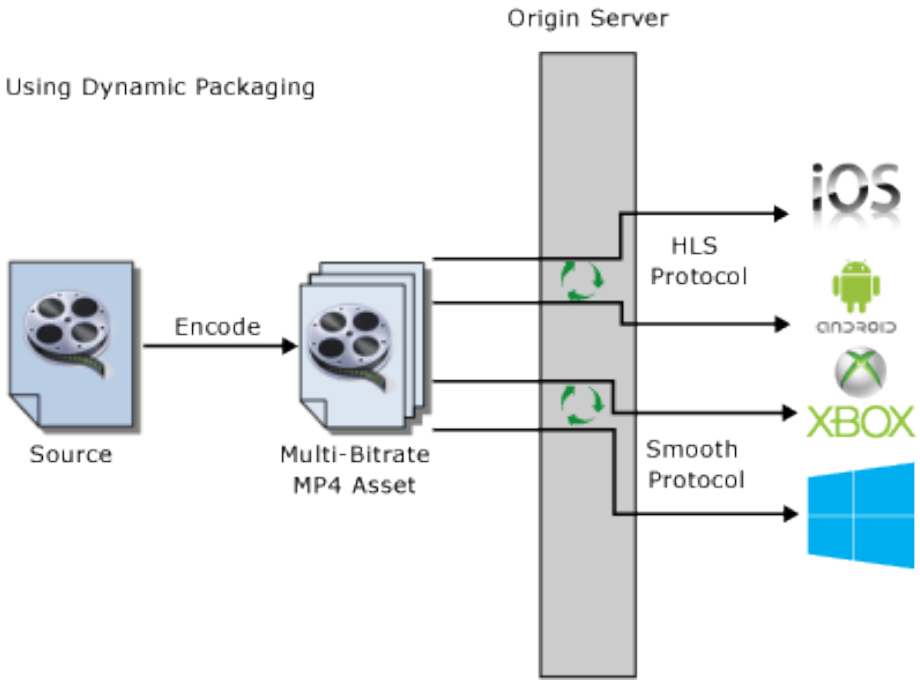
Progressiiviseksi lataukseksi kutsutaan menetelmää, jossa videota tallennetaan ensin käyttäjän levylle, jonka jälkeen sitä aletaan toistaa latauksen edetessä. Videon suoratoistossa puolestaan videon sisältöä ei tallenneta levylle, vaan video toistetaan suoraan ja sitä ladataan vain puskuriin. Käytettäessä progressiivista latausta videon lataus aloitetaan alusta ja videotiedostosta voidaan toistaa vain jo ladattu osa. Tästä seuraa se, että käyttäjä ei voi suoraan hypätä katsomaan tiettyä kohtaa videosta, ellei sitä ole ensin ladattu levylle. Videota suoraan toistettaessa käyttäjä voi siirtyä mihin kohtaan videota tahansa tai vaikka aloittaa katselun videon viimeisiltä minuuteilta. Kun käytetään suoratoistoa, palvelin osaa aloittaa videon lähettämisen videontoisto-ohjelman pyytämästä kohdasta. Progressiivista latausta käytettäessä käyttäjälle ei myöskään erikseen varata kaistaa, jotta videota voisi katsoa sujuvasti, vaan videon katsojamäärän kasvaessa myös lataus hidastuu ja vastaavasti puskurointi kestää kauemmin. [13]

Myös videomateriaalin katselumäärän seuraaminen on helpompaa toteuttaa suoratoiston yhteydessä. Progressiivisen latauksen tapauksessa videota ladataan käyttäjän levylle sillä nopeudella, jonka verkkoyhteys sallii. Esimerkiksi tunnin kestävä video saatetaan ehtiä lataamaan kokonaan, vaikka käyttäjä olisi katsonut videosta vain joitain minuutteja. Tällöin videon katselumäärää voidaan arvioida vain ladatun datan määrän perusteella mikä ei välttämättä anna luotettavaa kuvaa todellisuudesta. Käyttäjä voi lopettaa katselun joidenkin minuuttien jälkeen tai vastaavasti katsoa saman videon moneen kertaan. Käytettäessä videon suoratoistoa voidaan palvelimella puolestaan seurata ladatun datan määrän lisäksi suhteellisen luotettavasti myös katseluaikaa [18].

4.3.5 HTTP-pohjaiset suoratoistoprotokollat

Videon suoratoisto voidaan toteuttaa suoraan tätä tarkoitusta varten kehitetyllä protokollalla tai yhdessä HTTP-protokollan kanssa, jolloin tiedonsiirto tapahtuu HTTP-protokollalla. Tämän kaltaisia kaupallisia suoratoistoprotokollia ovat muiden muassa HTTP Live Streaming ja Microsoft Smooth Streaming.

HTTP Live Streaming on Applen iOS-käyttöjärjestelmässään käyttöönottama HTTP-pohjainen suoratoistoprotokolla. HTTP-protokollan käytöllä saavutetaan Jan Ozerin mukaan useita hyötyjä. Hän mainitsee muun muassa, että HTTP-protokollaa käytettäessä ei tarvita erillistä suoratoistopalvelinta. Lisäksi Ozerin mukaan on hyödyllistä, että palomuurit harvoin estävät HTTP-liikennettä. Microsoft Smooth Streaming on Microsoftin kehittämä adaptiivisen suoratoiston toteuttava videon suoratoistoprotokolla. [19, 20]



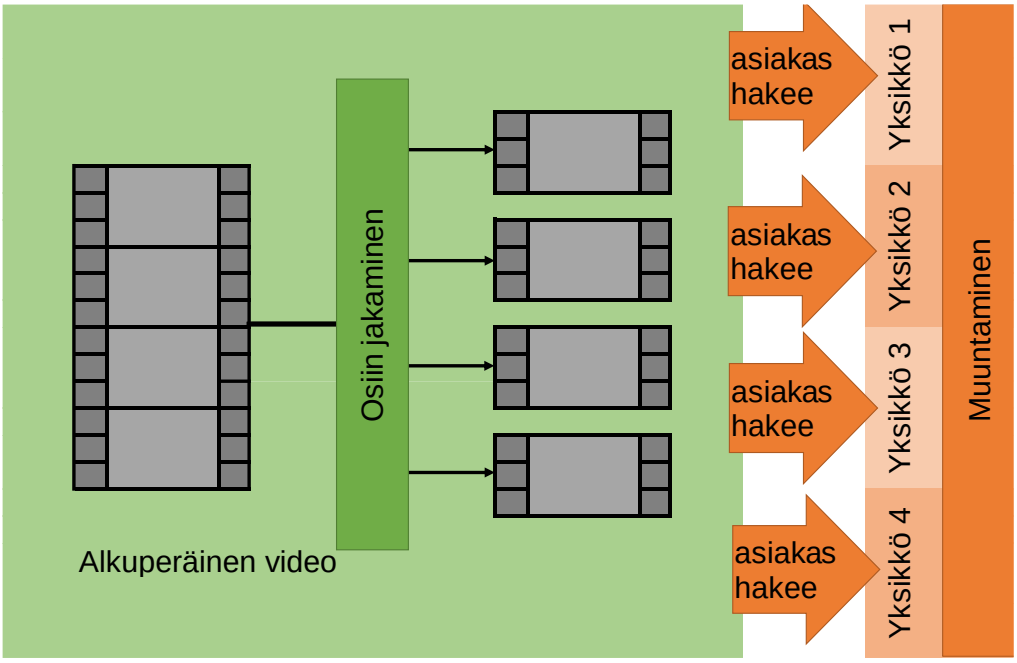
Kuva 4.3 *Suoratoisto adaptiivisella bittinopeudella [19].*

5 TESTIYMPÄRISTÖ

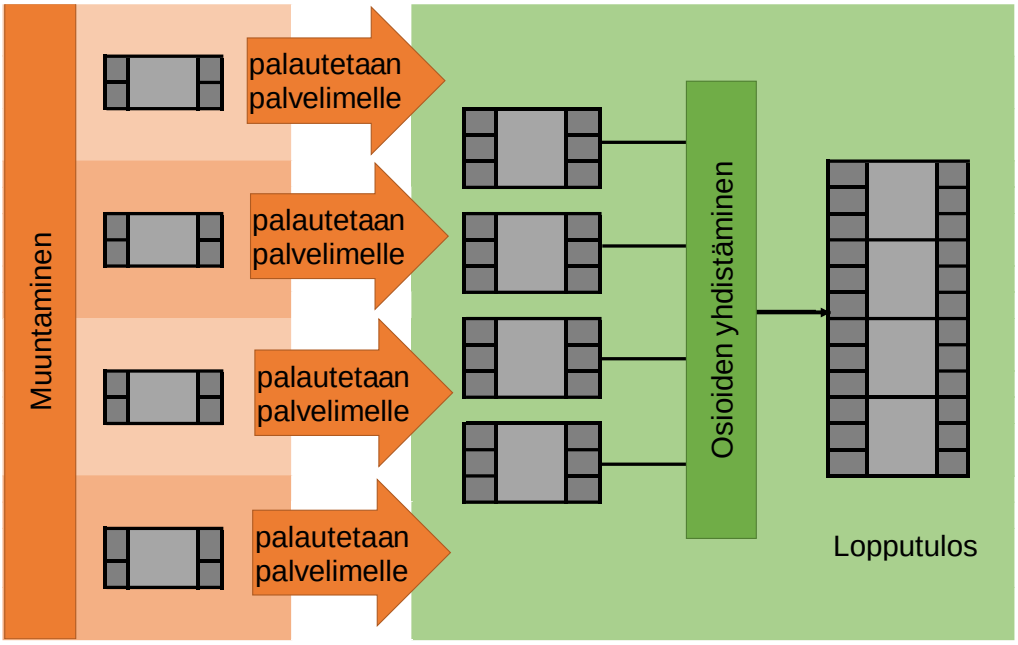
Videokoodausjärjestelmän skaalautuvuuteen liittyviä mittauksia varten toteutettiin testijärjestelmä, jonka avulla voidaan selvittää miten palveluna tarjottavat alustat soveltuvat videokoodauksen hajautettuun laskentaan. Tässä luvussa esitellään toteutetun järjestelmän osat ja rakenne, esitellään valittuja teknisiä ratkaisuja sekä kuvataan järjestelmän toiminnallisuutta.

5.1 Järjestelmän kuvaus

Testijärjestelmä koostuu verkossa toimivista tietokoneista, joista yksi on pääyksikkö ja loput laskentayksiköitä. Viestintä pää- ja laskentayksiköiden välillä toteutetaan pääasiassa käyttämällä viestijonoa; videotiedostojen siirtoon käytetään kuitenkin yhteistä verkkolevyä. Hajauttaminen perustuu videon osiointiin siten, että alkuperäisen videon osat lähetetään erillisinä tehtävinä laskettavaksi. Järjestelmän hajauttamisen periaate sekä laskentaprosessin eteneminen on kuvattu kuvassa 5.1.



Kuva 5.1a *Video jaetaan osiin, lähetetään laskettavaksi, palautetaan palvelimelle ja lopuksi yhdistetään.*



Kuva 5.1 b *Video jaetaan osiin, lähetetään laskettavaksi, palautetaan palvelimelle ja lopuksi yhdistetään.*

Kuvaan on piirretty järjestelmän eri osat:

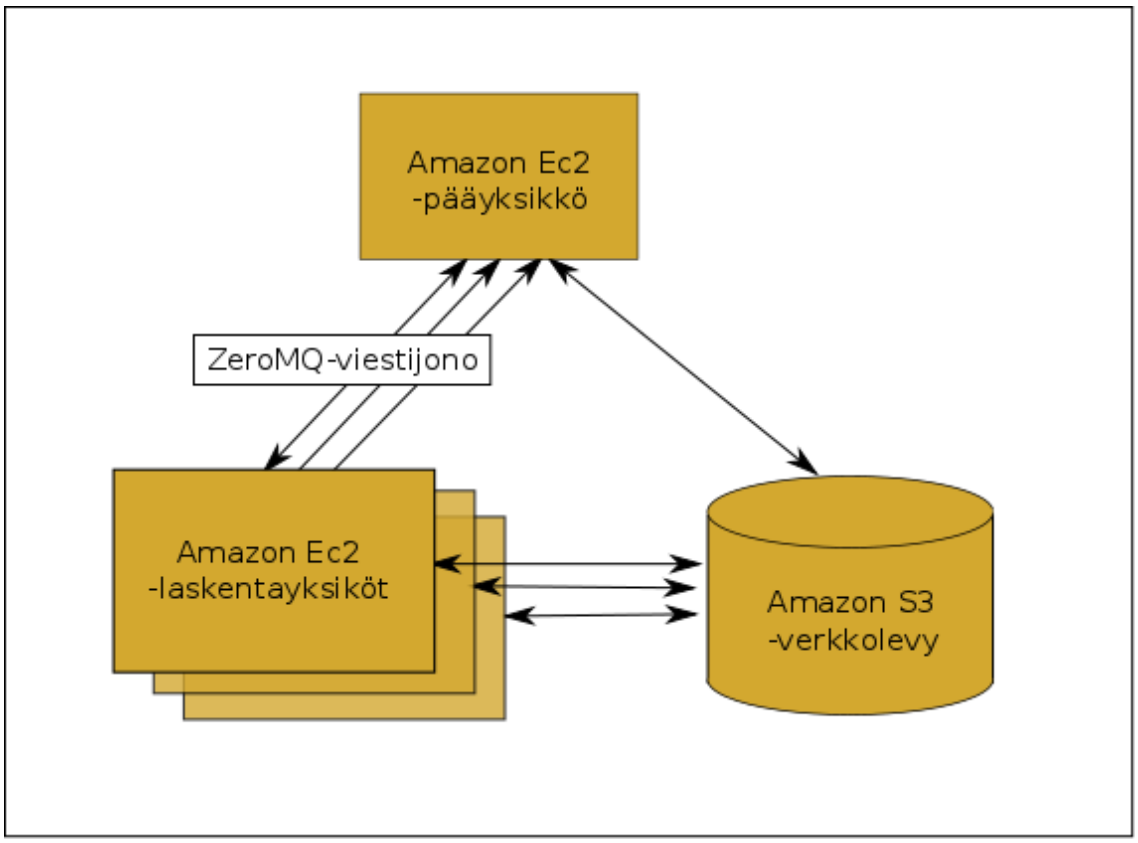
- palvelin- eli pääyksikkö (vihreällä taustalla)
- asiakas- eli laskentayksiköt (oranssilla taustalla)
- prosessin eteneminen nuolten suuntaan.

Palvelinkoneella tapahtuvat toiminnot ovat kuvattuna vihreällä taustalla. Kuvassa ylhäällä vasemmalla on alkuperäinen video, joka jaetaan osiin. Kuvassa on piirretty neljä osaa, mutta käytännössä näitä osioita on huomattavasti enemmän. H.264:llä koodattu video jaetaan ajallisesti sopivan kokoisiin osiin avainkehysten kohdalta, jolloin osiointi on nopea operaatio. Tällöin videolle ei tarvitse tehdä varsinaista muunnosta, koska juuri ennen avainkehystä tehty jako mahdollistaa uuden osion aloittamisen avainkehyksellä. Osat tallennetaan verkkolevylle, josta laskentayksiköt hakevat osaset palvelimelta saadun tiedon perusteella. Laskentayksiköt saavat siis tiedon uudesta työstä pyytämällä sitä palvelimelta, jolloin palvelin antaa kulloinkin vuorossa olevan osan laskentayksikölle laskettavaksi. Laskentayksikkö on kuvattu kuvassa oranssilla pohjalla.

Kun laskentayksikkö on hakenut osan alkuperäisestä videosta, se suorittaa koodimuunnoksen pyydettyyn videoformaattiin. Tämä koodimuunnos on prosessin työläin osa, ja sen kesto riippuu, paitsi laskentainstanssin tehosta, myös kohdetiedoston pituudesta, formaatista, resoluutiosta ja bittinopeudesta. Kun muunnos on suoritettu, tiedosto siirretään verkkolevylle, ja tieto suoritetusta tehtävästä lähetetään palvelimelle. Tämän pyynnön perusteella palvelin lataa tiedon verkkolevyltä. Kun videon kaikki osat on ladattu muunnettuna palvelimelle, palvelinkone yhdistää videot jälleen kokonaiseksi, jolloin lopputuloksena on alkuperäisestä videosta haluttuun formaattiin muunnettu video.

5.2 Järjestelmän osat

Toteutettu testiympäristö koostui Amazon Web Services –instansseista, jotka kommunikoivat keskenään. Yksi näistä toimii järjestelmän pääyksikkönä, joka kontrolloi prosessia. Pääyksikön vastuulla on videon jakaminen osiin, tallentaminen Amazon S3 -verkkolevylle sekä työtehtävien lähettäminen laskentayksiköille, valmiin videon osan hakeminen vastaavasti verkkolevyltä sekä videon yhdistäminen yhdeksi kokonaiseksi videoksi. Testijärjestelmässä ei toteutettu rajapintaa järjestelmän ulkopuolelle, vaan mittauksissa käytettävät tiedostot siirrettiin ensin pääyksikölle. Mittausten kannalta tämä ei ole oleellista, mutta tuotantojärjestelmässä tällainen rajapinta pitäisi luonnollisesti toteuttaa.



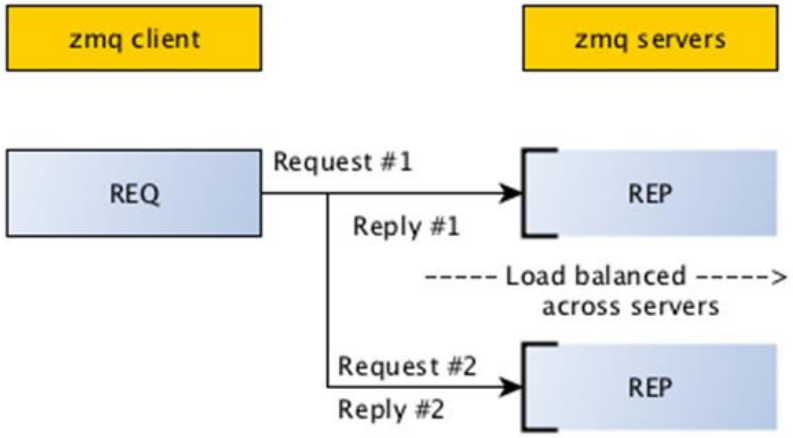
Kuva 5.2 *Järjestelmän osat: AWS pääyksikkö, yksi tai useampia laskentayksiköitä ja verkkolevy tietojen tallennusta varten.*

Pääyksikön lisäksi järjestelmään kuuluu yksi tai useampia Amazon EC2 -instansseja, jotka toimivat laskentayksiköinä, vastaanottavat laskentatehtävän viestinä, hakevat viestin ilmoittaman videon osan verkkolevyltä, tekevät määritellyn videomuunnoksen, tallentavat muunnetun videon verkkolevylle ja lähettävät laskennan valmistumisesta viestin pääyksikölle. Laskentayksiköt rekisteröityvät vastaanottamaan pääyksikölle, joka lähettää niille vuorotellen videopätkän muunnettavaksi. Järjestelmän osat on kuvattu kuvassa 5.2.

5.3 Viestien välitys ja tiedonsiirto

Viestintä instanssien välillä toteutettiin ZeroMQ-viestijonona lukuun ottamatta videotiedostojen siirtoa, joka toteutettiin tallentamalla videot verkkolevylle. ZeroMQ on asynkroniseen ja hajautettuun tiedonsiirtoon tarkoitettu sovellus, joka mahdollistaa eri alustojen ja ympäristöjen yhdistämisen toisiinsa. Yleisimmille ohjelmointikielille on toteutettu kirjasto ZeroMQ:n käyttämiseksi. [27] ZeroMQ soveltuu hyvin käytettäväksi toteutettavassa järjestelmässä, koska silloin ei tarvitse itse toteuttaa monimutkaista tiedonsiirtoa. Tällöin myös viestinnän rinnakkaisuuteen liittyvät seikat tulee otettua huomioon.

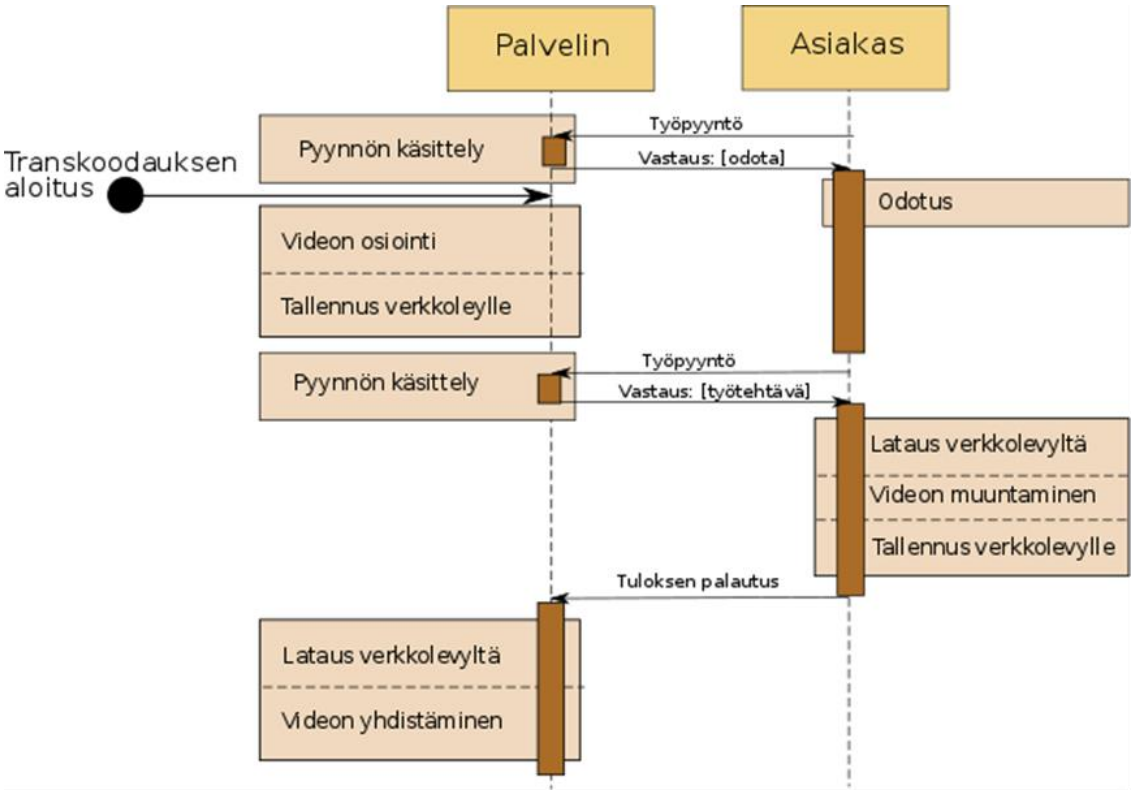
Viestijono toteutettiin käyttämällä asiakas/palvelin -tiedonsiirtomallia, jonka periaate on kuvattuna kuvassa 5.3. Palvelin asettaa viestijonoon tehtäviä, joita asiakasprosessit vuorollaan hakevat suoritettavaksi. Tämän jälkeen asiakas suorittaa viestissä määritellyn laskentatehtävän ja ilmoittaa siitä viestillä palvelimelle.



Kuva 5.3 *Asiakas/palvelin –tiedonsiirtomalli [21].*

Laskentayksikkö suorittaa kaikki yhteydenotot palvelimelle siitä syystä, että palvelinkoneen verkko-osoite voidaan asettaa staattiseksi, jolloin asiakaskoneet osaavat automaattisesti yhdistää ilman erillistä konfigurointia tai viestinvälitystä. Tämä yhteydenpito voisi tapahtua myös toisinpäin, jolloin palvelin ylläpitäisi tietoa käytössä olevista asiakkaista, ja jakaisi työtä sopivalla tavalla asiakkaille. Tämä kuitenkin vaatisi palvelimelta asiakaslistan ylläpitämisen lisäksi jonkinlaista kuormantasausalgoritmia. Tässä tapauksessa on kuitenkin vaikea kuvitella tehokkaampaa tapaa jakaa kuormaa, kuin että asiakkaat hakevat uuden työn heti vapauduttuaan edellisestä tehtävästä. Palvelinlähtöinen liikennöinti olisi hyödyksi esimerkiksi järjestelmän monitoroinnissa. Jotta monitorointi voitaisiin järkevästi esittää koko järjestelmän tasolta, olisi yksittäisten asiakaskoneiden tila joka tapauksessa välitettävä yhteen paikkaan. Tässä työssä ei kuitenkaan pyritä monitoroimaan yksittäisiä instansseja, vaan koko järjestelmää, jota sitäkin ainoastaan sen

suorittaman koodimuunnoksen tehokkuuden osalta. Koodimuunnosprosessin ajallinen eteneminen on esitettynä alla olevassa kuvassa 5.4.



Kuva 5.4 *Muunnoksen sekvenssikaavio.*

Kuvassa asiakassovellus lähettää määräajoin työpyynnön palvelimelle (kuva 5.5). Jos pavelimella ei ole valmiina suoritettavia tehtäviä, se vastaa pyyntöön odotusviestillä, joka on esitetty kuvassa 5.6. Tällöin asiakassovellus odottaa määrätyn ajan, jonka jälkeen työpyyntö lähetetään uudelleen.

```
{
   'type': 'job',
   'job': <job>,
   's3name': <s3filename>,
   'resolution': <resolution>,
   'bitrate': <bitrate>,
   'force_key_frames': <force_key_frames>
}
```

Kuva 5.5 *JSON-Työviesti*

Varsinainen prosessi alkaa siitä, kun käyttäjä käynnistää transkoodausprosessin palvelimella. Tämä on esitetty kuvassa 5.4 mustalla pallolla (transkoodauksen aloitus). Tällöin palvelinsovellus jakaa videon halutun kokoisiin osiin, lisää videon osat työjonoon

ja tallentaa ne verkkolevylle. Tämän jälkeen asiakkaalta tulevaan työpyyntöön vastataan lähettämällä tiedot muunnettavasta osiosta ottamalla työjonosta yksi osio muunnettavaksi. Työtehtävän vastaanotettuaan asiakas lataa videon verkkolevyltä, muuntaa sen tehtäväviestissä pyydettyyn muotoon, tallentaa takaisin verkkolevylle ja palauttaa tiedot muunnetusta videosta valmistumisilmoitusviestillä. Valmistumisilmoitus on esitetty kuvassa 5.7. Valmistumisilmoituksen saatuaan palvelin lataa osion verkkolevyltä paikalliselle levylle. Kun alkuperäisen videon kaikki osiot on muunnettu, yhdistetään video jälleen yhdeksi kokonaiseksi tulokseksi.

```
{'type': 'wait'}
```

Kuva 5.6 *JSON-odotusviesti*

```
{
        'type': 'result',
        'consumer' : <consumer_id>,
        'job': <job>,
         's3name': <result_bucket_name>
}
```

Kuva 5.7 *Valmistumisilmoitus*

Testijärjestelmässä järjestelmän palvelimena toimiva pääyksikkö on toteutettu käyttämällä ZeroMQ-viestijonon palvelin-socketia. Se toimii siten, että viestijonoon lisätään viestejä, jotka lähetetään lisäysjärjestyksessä pyyntöjen saapuessa asiakkailta. Asiakkaana toimivat laskentayksiköt siis saavat aina työjonossa olevan seuraavan viestin palvelimelta. Silloin kun jonossa on suoritettava työ, asiakas saa viestin ja jos työjonossa ei ole yhtään työtä, palauttaa palvelinprosessi odotuspyynnön.

5.4 Instanssien hallinta

Palvelinkoneita on vain yksi, joka on testitapauksia suoritettaessa jatkuvasti päällä. Palvelinkoneella oleva ohjelma käynnistää kulloinkin halutun määrän asiakaskoneita, jotka käynnistyttyään hakevat palvelimelta jonossa olevan olevia laskentatehtäviä. Mittauksia suoritettaessa tosin laskentayksiköt käynnistettiin etukäteen, jolloin kuormantasausta ei tarvinnut tehdä palvelinkoneella. Tästä olisi hyötyä vain ajettaessa järjestelmää tuotannossa. Laskentatehomittausten kannalta palvelimella oleva kuormantasaus käytännössä vain mutkistaisi tulosten arviointia. Tietenkin mitattaessa järjestelmän kykyä sopeutua muuttuvaan kuormitukseen on myös mittauksissa otettava huomioon järjestelmän automaattinen skaalautuminen.

Molemmat koneet ovat samanlaisia Amazon EC2 -instansseja ja eroavat keskenään vain ajettavan ohjelmiston osalta. Testejä ajetaan palvelinkoneella komentoriviltä,

johon yhteys otetaan SSH:lla. Asiakaskoneelle ei oteta yhteyttä, vaan siihen asennettu asiakasohjelmisto ottaa käynnistyttyään yhteyden palvelimelle. Palvelinohjelma vastaa myös asiakaskoneiden sammuttamisesta, joskin testattaessa on hyödyllistä hoitaa tämä käsin AWS:n käyttöliittymästä. Palvelinohjelma hallinnoi asiakaskoneita AWS-ohjelmistorajapinnan kautta. Tämän työn kannalta tarpeellista on vain uusien instanssien käynnistys ja sammuttaminen.

Käytännön syistä kuitenkin instanssien käynnistäminen hoidettiin käsin, koska ajossa olevista instansseista laskutetaan tuntikohtaisesti. Käytännössä siis minimilaskutus on yhden tunnin hinta, vaikka instanssi olisi todellisuudessa ollut ajossa vain joitain minuutteja. Lisäksi testauksessa oli parhaillaan käytössä 150 instanssia, joten niiden useasti toistuva käynnistäminen ja sammuttaminen olisi lisännyt kustannuksia selvästi. Testattaessa instanssit siis käynnistettiin testisession alussa ja sammutettiin lopussa, jolloin turhilta kustannuksilta säästyttiin. Tuotantojärjestelmässä olisi syytä olla instanssien riittävän kehittynyt algoritmi, joka ottaisi myös tämän huomioon. Laskentainstanssien automaattisesta sammuttamisesta voisi vastata joko pääinstanssi tai laskentainstanssi itse. Laskentainstanssi voisi sammuttaa itsensä silloin, kun työtä ei ole ollut pitkään aikaan ja instanssi on ollut päällä sopivan ajan. Käynnistyksestä vastuu on kuitenkin oltava pääinstanssilla.

5.5 Koodimuunnos

Videon koodimuunnos suoritetaan palvelimelta saatavassa työviestissä ilmoitetuilla asetuksilla. Koodimuunnos suoritetaan Linux-ympäristössä helposti saatavilla olevalla FFmpeg-ohjelmalla, joka soveltuu tähän tarkoitukseen hyvin. Videokoodauksessa käytettäviä parametreja on koko järjestelmän laajuudelta määritelty vain kolme: resoluutio, bittinopeus ja avainkehyksen asettaminen tasaväliseksi (force_key_frames). Tässä työssä tämä ei kuitenkaan haittaa, koska tarkoitus ei ole tutkia koodimuunnoksen toimivuutta muuten kuin niiltä osin, mitä vaaditaan rinnakkaistumisen ja tehokkuuden mittaamiseen. Riittää siis, että osiin jaettu video voidaan jälkikäteen jälleen yhdistää uudeksi videoksi, ja että mitattavat ajat riittävässä määrin vastaavat todellisia tapauksia. Tältä osin tärkeää on myös, että videon laskennan hajauttaminen voidaan toteuttaa oikeasti, jolloin kyseisenkaltainen järjestelmä voidaan todeta toimivaksi ja toteutuskelpoiseksi.

5.6 Ongelman rajaus

Työssä rajoituttiin tarkastelemaan yhden videon muunnosta kerrallaan. Lopullisessa tuotantototeutuksessa olisi otettava huomioon monen videon yhtäaikainen muuntaminen, varauduttava hetkittäiseen suureen kuormitukseen ja, jotta järjestelmän dynaamisesta skaalauksesta olisi hyötyä, olisi jotenkin voitava ennustaa tulevaa kuormaa esimerkiksi vuorokauden ajan tai viikonpäivän mukaan.

Myöskään videon laatuun ei kiinnitetty suurta huomiota. Tärkeää tältä osin oli, että video on jaettavissa osiin ja lopputulos on järkevä. Testijärjestelmästä ei pyritty saamaan videoformaatin suhteen kattavaa tai robustia, vaan riittää, kun löydetään jokin formaatti, jolla tätä voidaan soveltaa. Tämä ei välttämättä riitä jokaiseen sovelluskohteeseen.

6 MITTAUS

Edellisessä luvussa esiteltiin transkoodausjärjestelmän testiympäristön toiminta ja suunnitteluprosessi. Tässä luvussa on tarkoitus esittää suoritettavat mittaukset, jotka on pyritty suunnittelemaan sellaisiksi, että ne toisivat mahdollisimman hyvin esille sen, miten edellisessä luvussa kuvattu järjestelmä soveltuu videokoodimuunnoksen hajautettuun laskentaan.

Amazon AWS -pilvipalvelinympäristö tarjoaa hyvät työkalut palvelinalustojen dynaamiselle käyttöönotolle, ja samalla järjestelmän skaalaamiselle. Hyvin onnistuneiden mittausten avulla saadaan selville, miten käyttökelpoinen kyseinen julkinen ympäristö on tällaiseen tarkoitukseen. Alaluvussa 6.1 esitellään ja perustellaan testiaineiston eli alkuperäisen videon valintaa sekä alaluvuissa 6.2 – 6.3 kuvataan suoritettuja mittauksia. Alaluvussa 6.2 esitellään laskentayksiköiden määrän vaikutusta järjestelmän tehokkuuteen ja alaluvussa 6.3 mitataan muunnoksen eri vaiheissa kuluvaa aikaa.

6.1 Testiaineisto

Tehdyt mittaukset suoritettiin käyttämällä avoimen lähdekoodin projektia Big Buck Bunny, joka julkaistiin vuonna 2008 (kuva 6.1). Tämä video valittiin pääasiassa siksi, että se on helposti saatavilla ja vapaasti levitettävissä. Tästä syystä se oli luonteva valinta tässä työssä tehtyihin mittauksiin. Lisäksi se on vapaasti saatavilla myös mahdollista jatkotutkimusta varten. Kyseinen video on myös pituudeltaan sopiva jaettavaksi osiin. Samoin tällä testivideolla voidaan mitata järjestelmän tehokkuutta ja silti suorittaa monta testiajoa lyhyessä ajassa.

Testivideo on valmiiksi muunnettu vähemmän tilaa vievään muotoon, jotta tiedonsiirto veisi järkevän ajan mittauksessa. Muunnettavat videot ovat todennäköisesti kooltaan suurempia, ja kuormittavat järjestelmää enemmän myös muunnoksen osalta. Todellisuudessa jouduttaisiin laskennassa käyttämään tehokkaampia tietokoneita, mutta rajallisista kustannuksista johtuen näissä mittauksissa jouduttiin käyttämään halvinta käytettävissä olevaa instanssityyppiä: EC2 nano -instanssi. Tässä instanssityypissä oleelliset rajoitukset ovat suoritinteho, verkon suorituskyky ja keskusmuistin määrä. Jotta mittaus toisi esiin järjestelmän tehokkuusominaisuudet, myös testivideon laatua on heikennetty. Tällä tavalla saadaan sekä tiedonsiirrosta että laskennassa käytetty aika hieman pienemmäksi kuitenkin siten, että se vastaisi mahdollisimman hyvin todellista tilannetta. Testivideon pituus on 9 min 56 s, resoluutio on 400x224 ja tiedoston koko on 24,2 MiB.

Kuva 6.1 Kuvakaappaus testivideosta Big Buck Bunny. [24]

Kaikki tässä työssä suoritettavat mittaukset käyttäytyvät oletettavasti samalla tavalla millä tahansa videolla, joten yksi video riittää käytettäväksi mittauksissa.

6.2 Laskentayksiköiden määrä

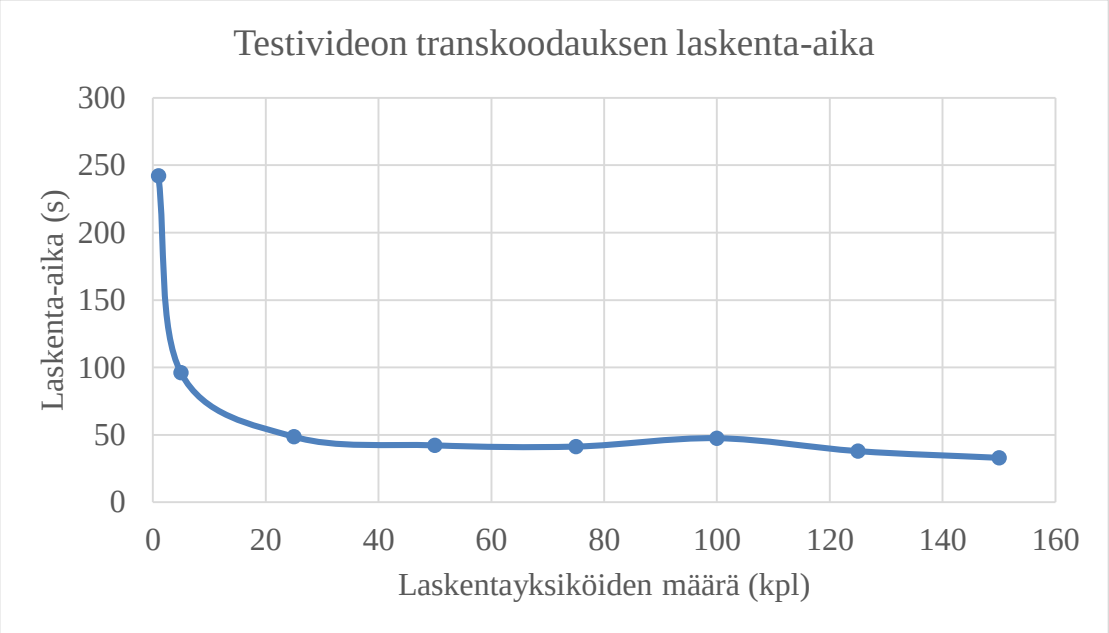
Yksi helpoimmista tavoista lisätä järjestelmän suorituskykyä on lisätä laskentayksiköiden määrää. Tätä ei voida kuitenkaan tehdä loputtomasti, vaan jossain vaiheessa järjestelmän kokonaislaskentatehon lisääminen ei enää kannata. Tämä johtuu sekä siitä, että kaikkea työtä ei voida hajauttaa että hajauttamiseen liittyvästä muusta tarpeesta kuten tiedonsiirrosta. Jotta videokoodausta voidaan hajauttaa se pitää voida jakaa erillisiin muunnettaviin osiin. Lisäksi jokainen osa on lähetettävä laskentayksikölle, joka vaatii tiedonsiirtoa. Mitä enemmän työtä rinnakkaistetaan, sitä suurempi hetkellinen kuormitus tästä aiheutuu tietoverkolle.

Tässä mittauksessa on tavoitteena selvittää miten laskentayksiköiden määrä vaikuttaa koko järjestelmän suorituskykyyn. Instanssimäärän vaikutus pyrittiin saamaan esille valitsemalla mittauksen kohteeksi seuraavat instanssimäärät (kpl): 1, 5, 25, 50, 75, 100, 125 ja 150. Instanssimäärän lisäys tehtiin pääasiassa 25 yksikköä kerrallaan. Tämän lisäksi mittaus tehtiin myös viidellä ja kahdellakymmenelläviidellä yksiköllä, jotta suhteellinen lisäys ei olisi liian suuri alussa. Tällöin lisäämisen vaikutus pienillä instanssimäärillä saadaan selvitettyä tarkemmin. Ylärajaksi valittiin 150, koska oli oletettavaa, että lisäyksestä aiheutuva vaikutus on nähtävissä jo sitä pienemmillä määrillä. Taulukko 6.1 sisältää mitatut ajat sekunteina. Taulukossa on nähtävillä käytetty instanssimäärä, laskentavaiheeseen kulunut aika ja muunnoksen kokonaisaika. Instanssimäärän ollessa pieni uuden instanssin lisääminen hyödyttää enemmän kuin silloin, kun instanssimäärä on suuri.

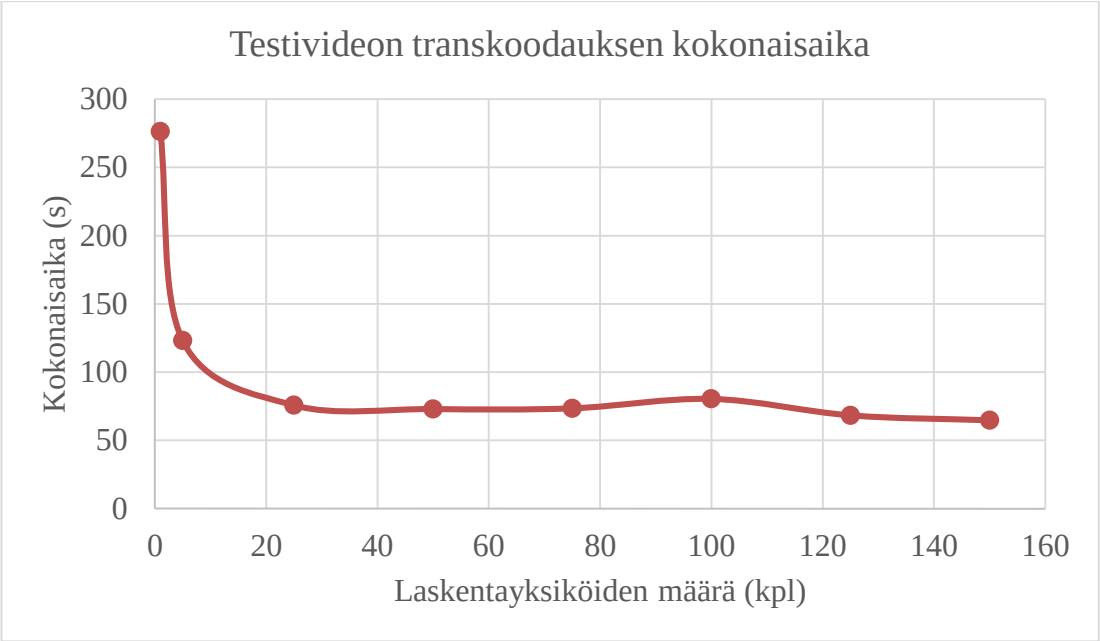
Taulukko 6.1 *Laskenta- ja kokonaisajat*

Instanssien lukumäärä	Laskentaan kulunut aika (s)	Kokonaisaika (s)
1	242	276
5	96	123
25	49	76
50	42	73
75	41	74
100	48	80
125	38	68
150	33	65

Seuraavat kuvaajat esittävät laskenta-ajan kehitystä, kun laskentayksiköiden määrää kasvatetaan. Kuva 6.2 kuvaa itse laskentaan kuluvaa aikaa laskentayksiköiden määrän funktiona ja vastaavasti kuva 6.3 kuvaa koko järjestelmän suoritusaikaa. Kuvaajat tuovat hyvin esille suorituskyvyn kasvun pienillä instanssimäärillä. Tehonlisäys pienenee kuitenkin merkittävästi instanssimäärän kasvaessa, mikä näkyy käyrän suhteellisen pienenä kulmakertoimena, kun instanssimäärä ylittää arvon 30.



Kuva 6.2 *Testivideon transkoodaukseen käytetty laskenta-aika eri laskentayksikkömäärillä.*



Kuva 6.3 *Testivideon transkoodaukseen käytetty kokonaisaika eri laskentayksiköiden määrillä.*

Kahdessa edellisessä kuvaajassa on erona se, että jälkimmäisessä on mukana kaikki järjestelmän vaiheiden kestot, kun aiemmassa on vain laskentaan kulunut aika. Prosessin muiden osien kestot ja laskennan suhde kokonaisuuteen on tarkasteltavana seuraavassa alaluvussa.

6.3 Laskennan vaiheet

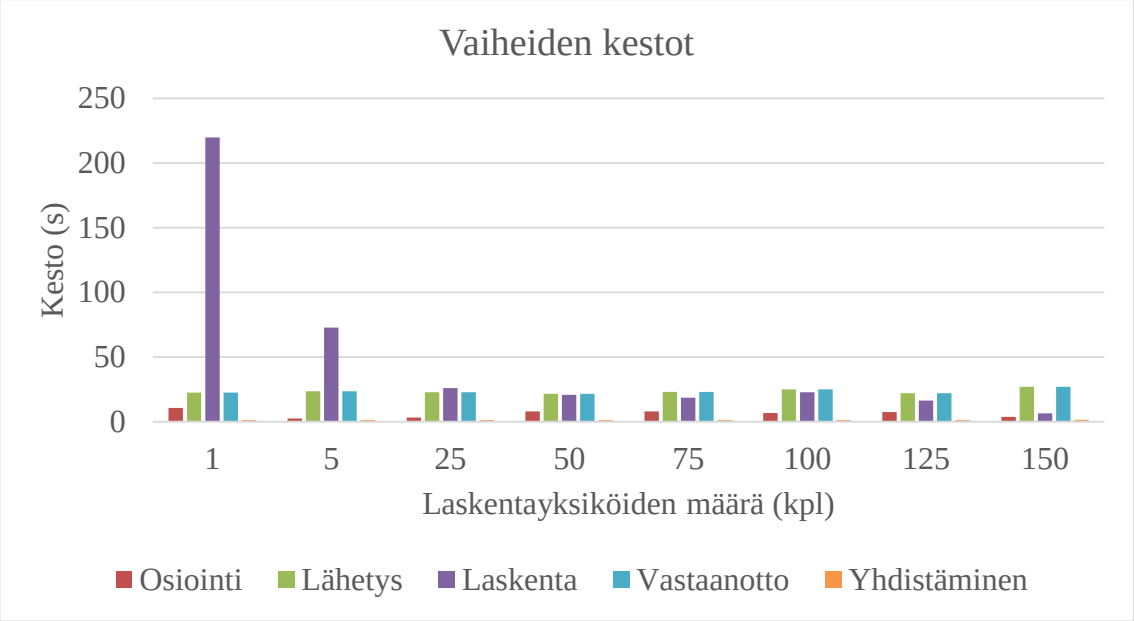
Pilvilaskennassa aikaa kuluu myös muuhun kuin itse laskentaongelmaan. Työ voidaan jakaa eri vaiheisiin esimerkiksi seuraavasti:

- osiointi
- tehtävän lähetys
- laskenta
- tuloksen lähetys
- yhdistäminen.

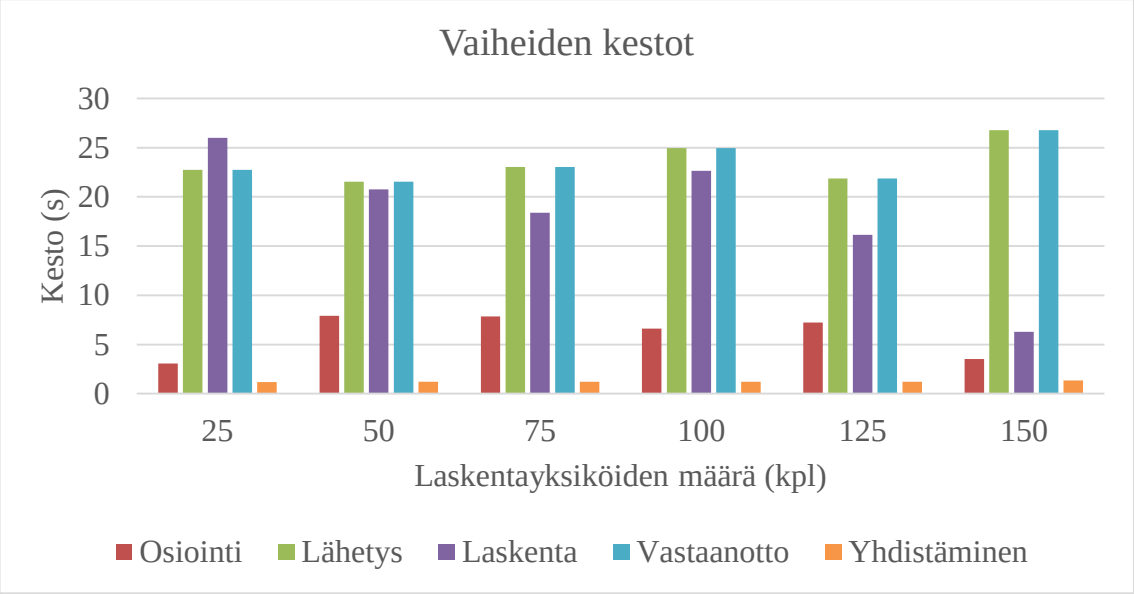
Transkoodausprosessin ajoituksista voidaan analysoida eri vaiheisiin kuluvat ajat. Osiointi- ja yhdistämisprosesseihin kuluva aika riippuu paitsi tiedoston koosta, myös tiedostoformaatista ja käytettävästä osiointialgoritmista.

Tiedonsiirron merkitys kokonaisprosessissa kasvaa, mitä isommaksi instanssimäärää kasvatetaan. Vaikka tiedonsiirtoa joudutaan tekemään saman verran riippumatta hajautusasteesta, se muuttuu merkittäväksi osaksi kokonaisuutta. Mitä enemmän työtä halutaan rinnakkaistaa, sitä useamman laskentayksikön pitäisi päästä suorittamaan laskentatehtävää mahdollisimman ajoissa. Tästä aiheutuu se, että usealle yksikölle rinnakkaistettaessa joudutaan koko videosta suurempi osa lähettämään samanaikaisesti laskentayksiköille. Mitä useammalle instanssille yritetään lähettää työtä samanaikaisesti, sitä

enemmän vaaditaan tiedonsiirtokapasiteettia ja sitä merkittävämmäksi sen osuus kokonaisuudesta muodostuu. Tähän vaikuttaa myös tiedoston koko: mikäli tiedosto on valmiiksi pieni, ei siitä aiheudu merkittävää tiedonsiirtoa. Tämän kaltaisessa tapauksessa todennäköisesti myös laskentatyö on yleensä nopeampi, joten tiedonsiirron osuus kokonaiskestosta ei todennäköisesti pienene merkittävästi.



Kuva 6.4 *Eri vaiheiden kestot mitatuilla laskentayksiköiden määrillä.*



Kuva 6.5 *Eri vaiheiden kestot 25–150 laskentayksiköllä.*

Yllä olevissa kuvaajissa on kuvattuna eri vaiheisiin kuluvat ajat, kun laskentayksiköiden määrää kasvatetaan. Ylemmässä kuvassa 6.4 on mukana ajoitukset kaikilla eri

instanssimäärillä. Tämä osoittaa hyvin, että laskentavaiheessa kuluva aika pienenee merkittävästi pienillä instanssimäärillä, ja myös sen, että laskentavaiheen osuus yhdellä instanssilla laskettaessa vie käytännössä kaiken prosessiin kuluvan ajan, kun taas 150 yksiköllä sen osuus koko kestosta jää lähes merkityksettömäksi. Alkuosia lukuun ottamatta vaiheiden kestot jakautuvat suunnilleen samalla tavalla. Tämä on helppo huomata alemmasta kuvasta, jossa alkupään pylväät on jätetty pois paremman luettavuuden saavuttamiseksi.

7 ARVIOINTI

Tässä luvussa arvioidaan edellä esitettyjä mittaustuloksia. Tähän sisältyy laskentayksiköisen määrään liittyvien mittausten sekä järjestelmän tiedonsiirron tehokkuuden arviointia. Lisäksi arvioidaan järjestelmään liittyviä kustannuksia sekä pohditaan järjestelmän kannattavuutta ja optimointia.

7.1 Mittaustulosten arviointi

Tässä luvussa analysoidaan edellisessä luvussa esiteltyjä mittaustuloksia. Tavoitteena on arvioida mittauksien mielekkyyttä ja mittausten perusteella tehdä päätelmiä videotranskoodauksen hajauttamisen onnistumisesta ja tehokkuudesta.

7.1.1 Laskentayksiköiden määrä

Koejärjestelyssä laskentayksiköiden määrän kasvattaminen lisäsi kokonaistehokkuutta noin 25 yksikköön asti. Tätä useammalla laskentayksiköllä instanssimäärän kasvattamisesta saatava hyöty oli vähäistä, sillä kokonaislaskenta-aika lyheni sitä vähemmän mitä enemmän rinnakkaisia laskentayksiköitä oli alun perin käytössä. Yli 50 yksiköllä vaikutus laskentateho ei lisääntynyt käytännöllisesti katsoen ollenkaan. Kun instanssien määrää kasvatettiin viisinkertaiseksi yhdestä viiteen, laskenta-aika nopeutui 60 prosenttia. Kun taas määrä viisinkertaistettiin välillä 25–125, laskenta nopeutui enää 22 prosenttia. Kokonaisaikaa mitattaessa laskentayksiköiden lisääminen nopeutti järjestelmää 55 prosenttia lisättäessä määrää yhdestä viiteen ja 10 prosenttia, kun määrää lisättiin 25:stä 125:een.

Koejärjestelyn laajuus instanssimäärän osalta riitti tuomaan esille sen, että laskentateho pienenee kasvatettaessa instanssimäärää. Ylärajaksi valittu 150 kpl riitti hyvin tuomaan esille sen, että suurilla instanssimäärillä ei saavutettu enää merkittävää hyötyä. Valittu mittausskaala oli riittävän suuri tämän työn kannalta. Voidaan olettaa, että mitä vaativampi työ on suhteessa siirrettävän tiedoston kokoon, sitä suurempi hyöty saadaan rinnakkaisesta laskennasta. Tämä johtuu siitä, että pienikokoisen tiedoston siirtäminen kuormittaa vähän verkkoa suhteessa laskenta-aikaan. Isokokoinen ja nopeasti laskettava tiedosto puolestaan kannattaa laskea paikallisesti.

Edellä kuvastusta tiedonsiirtovaatimuksesta johtuen oli jo ennen mittauksia oletettavissa, että hyöty pienenee suhteessa rinnakkaisten instanssien määrään. Mitä useammalle laskentayksiköllä työ hajautetaan, sitä suurempi on hetkellinen kuormitus tiedonsiirrossa sekä lähetettäessä että vastaanotettaessa. Jos kaikki tiedoston osat lähetetään samanaikaisesti eri instansseille, se kuormittaa hetkellisesti huomattavasti enemmän verkkoa kuin yksi kerrallaan lähettäminen. Toinen syy sille, että hajauttamisesta saatava hyöty pienenee, mitä enemmän rinnakkaistetaan, on laskentavaiheen keston osuuden pieneneminen koko prosessin kestosta. Laskentaa rinnakkaistettaessa ainoastaan laskentavaiheen

osuus pienenee muiden pysyessä, pientä vaihtelua lukuun ottamatta, vakiona. Muita prosessin osia ei voida yhtä helposti rinnakkaistaa, joten edes teoriassa kokonaisaika ei pienene suoraan suhteessa laskentainstanssien määrään. Kuten aiemmin todettiin, mitä intensiivisempää laskentaa muunnoksessa vaaditaan, sitä suurempi hyöty saavutetaan rinnakkaistamisesta. Rinnakkaistamisesta aiheutuvan hyödyn teoreettinen maksimi saavutetaan silloin, kun yksiköitä on yhtä monta kuin jaetun videon osioita. Tällöin jokainen osio voidaan lähettää suoraan vapaalle laskentayksikölle, eikä aikaa laskentayksikön vapautumiseen kulu ollenkaan.

Niin kuin edellä kuvattiin, ainoastaan laskentavaihe voitiin hajauttaa ja rinnakkaistaa erillisille instansseille. Tämä tulee myös mittauksissa esille laskentavaiheiden keston vertailussa. Muiden vaiheiden kestot olivat myös mitattaessa vakiosuuruiset. Kokonaisaika nopeutuu suhteellisesti vähemmän kuin laskenta-aika.

Laskentayksiköiden lisäämisestä aiheutuva hyöty jäi kuviteltua vähäisemmäksi, vaikka käytössä olivat vähiten tehokkaat saatavilla olleet, Amazonin tarjoamat, instanssit. Koko järjestelmän tehonlisäys alkoi olla vähäistä noin kolmenkymmenen laskentayksikön lisäyksen jälkeen. Tämä kertoo siitä, että koko järjestelmän nopeus riippuu käytettävissä olevasta teoreettisesta laskentatehosta vain pienillä instanssimäärillä.

7.1.2 Tiedonsiirto

Toteutetussa mittauksessa tiedonsiirto aiheutti suurimmat toteutukseen liittyvät ongelmat. Se myös tuli ensimmäisenä vastaan järjestelmää skaalattaessa. Työssä tiedonsiirto toteutettiin aikataulullisista syistä käyttämällä Amazonin S3-järjestelmää, jossa tiedostot tallennetaan ulkoiselle verkkolevylle. Tällöin tiedosto on käytännössä siirrettävä kahteen kertaan, koska laskentayksikkö voi hakea tiedon vasta, kun se on tallennettu S3-verkkolevylle ja viesti tiedoston tallentamisesta on edelleen lähetetty kyseiselle yksikölle. Tästä aiheutuu turhaa verkkoliikennettä, joka voidaan ratkaista joko lisäämällä tiedonsiirtokapasiteettia tai käyttämällä tehokkaampaa menetelmää tiedon siirtämiseen. Tehokkaampi tapa tiedonsiirtoon olisi esimerkiksi tiedoston lähettäminen suoraan laskentainstanssille sen vapautuessa käyttöön. Tämä hyödyttäisi kahdella tavalla: tiedostot siirretään suoraan lähettäjän ja vastaanottajan välillä sekä tiedonsiirto jakautuu ajallisesti tasaisemmin laskentatyön kanssa. Näin säästettäisiin sekä tiedonsiirron määrässä että verkon hetkittäisessä kuormituksessa.

Mikäli lähetettävä data olisi siirretty samassa viestissä laskentatehtävän mukana, voidaan olettaa, että järjestelmää oltaisi voitu skaalata jonkin verran enemmän. Tästä aiheutuva hyöty on sitä merkittävämpi, mitä vähemmän kapasiteettia tiedonsiirtoon on käytettävissä.

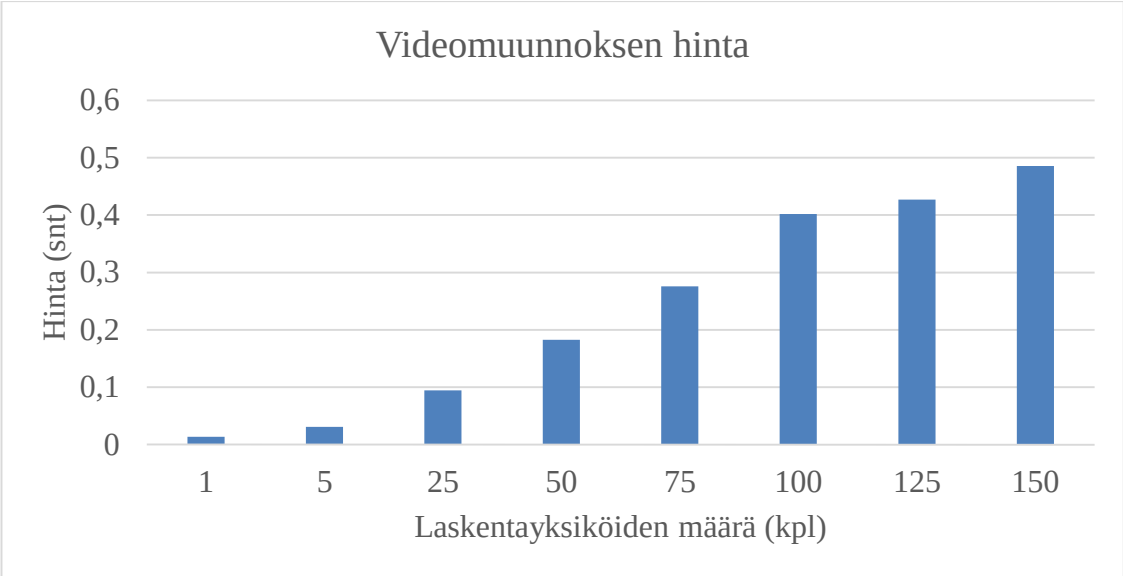
7.2 Hinta ja kannattavuus

Videokoodauksen hajauttamisesta saavutettavaan hyötyyn liittyy edellä käsitellyn, suorituskykyvaatimuksiin liittyvän, näkökulman lisäksi toinenkin merkittävä näkökulma: hajautuksesta aiheutuvat rahalliset kustannukset.

7.2.1 Laskennan kustannusarvio

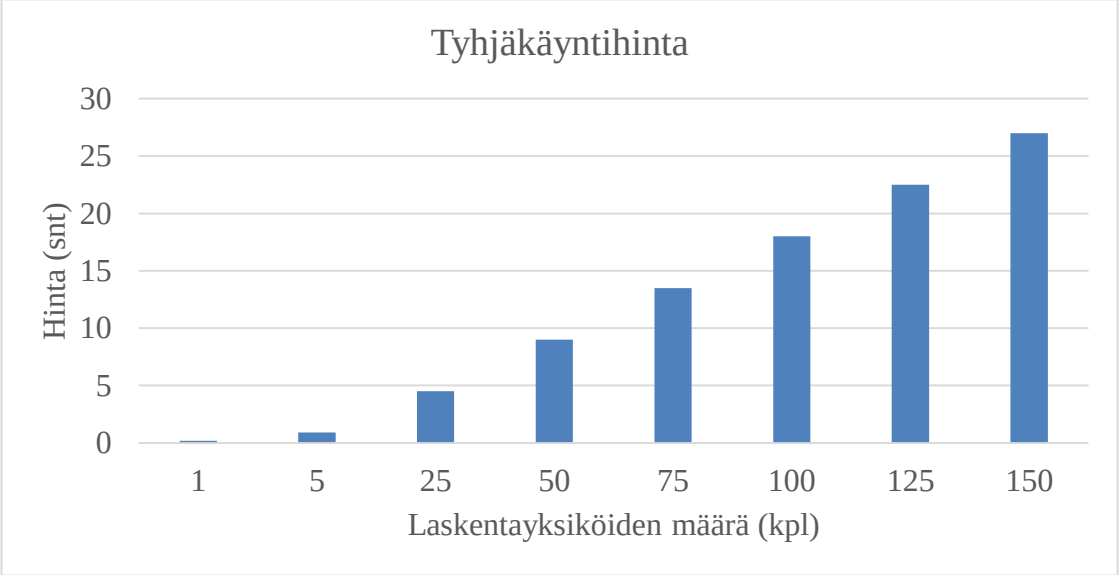
Videokoodauksesta aiheutuvat kustannukset riippuvat monesta tekijästä. Perinteisesti, kun videomuunnos suoritetaan yhden tietokoneen ympäristössä, laskentaan vaikuttavat kustannukset muodostuvat pääasiassa laitteiston ohjelmiston hankinnasta, asennuksesta ja ylläpidosta. Jos laskentaa haluttaisiin hajauttaa usealle tietokoneelle, jouduttaisiin nämä hankkimaan erikseen, jolloin kustannus syntysi hankintavaiheessa. Kun taas käytetään julkisia pilvipalvelualustoja, laitteisto ja ohjelmistot kuuluvat osana palveluna tarjottavaan alustaan eikä kustannus lisäänny, vaikka laskentaa hajautettaisiin usealle tietokoneelle kerrallaan. Pilvipalveluiden tapauksessa laskennan kustannukseen vaikuttaa ainoastaan laskentaan käytettyjen instanssien ajoaika. Hinta ei siis riipu siitä, kuinka monta tietokonetta on kerrallaan ajossa vaan siitä, kuinka pitkä on niiden yhteenlaskettu käyttöaika. Koska laskentaan käytettävien yksiköiden hinnoittelu on tuntiperusteinen, voidaan hajautetun laskennan hintaa kuvata vastaavasti tuntihinnan ja käytetyn ajan suhteena, jolloin saadaan videon muunnoksen hinta. Mittauksissa käytetyn Amazon EC2 nano -instanssin hinta on kirjoitushetkellä 0,18 snt/h.

Kuva 7.1 esittää videomuunnoksen hinnan kasvun, kun laskentayksikkömäärää kasvatetaan. Kun otetaan huomioon vain laskennassa kulunut aika, tulee hinnaksi yhdellä yksiköllä 0,01 senttiä. Vastaavasti 150 instanssilla hinnaksi tulee n. 0,5 senttiä. Kuvasta voidaan päätellä, että mitä enemmän laskentayksiköitä laskennassa käytetään, sitä suurempi kustannus tulee, kun verrataan saman videon koodimuunnokseen kuluvaa aikaa. Koska hajautuksesta saavutettava hyöty vähenee instanssimäärää kasvatettaessa, on laskenta, puhtaasti kustannusmielessä ajateltuna, tehokkainta suorittaa yhdellä tietokoneella paikallisesti. Tällöin kulunut aika on kuitenkin merkittävästi isompi kuin hajautettaessa. Jos taas kustannuksia ei oteta huomioon, olisi puolestaan laskenta-ajan kannalta tehokkainta suorittaa laskenta mahdollisimman monella instanssilla. Valittaessa järjestelmän kokonaiskapasiteettia on siis mietittävä, mitkä järjestelmän ominaisuudet ovat tärkeimpiä ja mistä voidaan tinkiä. On löydettävä jokin tasapaino, jolloin järjestelmä toteuttaa halutut ominaisuudet mahdollisimman hyvin.



Kuva 7.1 *Videomuunnoksen hinta verrattuna laskentayksiköiden määrään.*

Kuvassa 7.1 esitetyissä hinnoissa on oletettu, että instanssi on ajossa vain laskentaprosessin aikana, joten kuvaajan hinnat sisältävät vain todellisen laskentaan kulutetun ajan. Todellisessa järjestelmässä instanssien käynnissäoloaikaa ei voida optimoida näin hyvin, vaan instansseille tulee väkisinkin jonkin verran odottelua. Odotusaika saataisiin vähäiseksi, jos instanssien laskutus olisi joustavampi kuin Amazonilla käytössä ollut tapa, jossa käynnissäoloaika lasketaan tunnin tarkkuudella. Instanssista laskutetaan aina alkavalta tunnilta, joten vähäisestäkin käytöstä laskutetaan vähintään tunti.



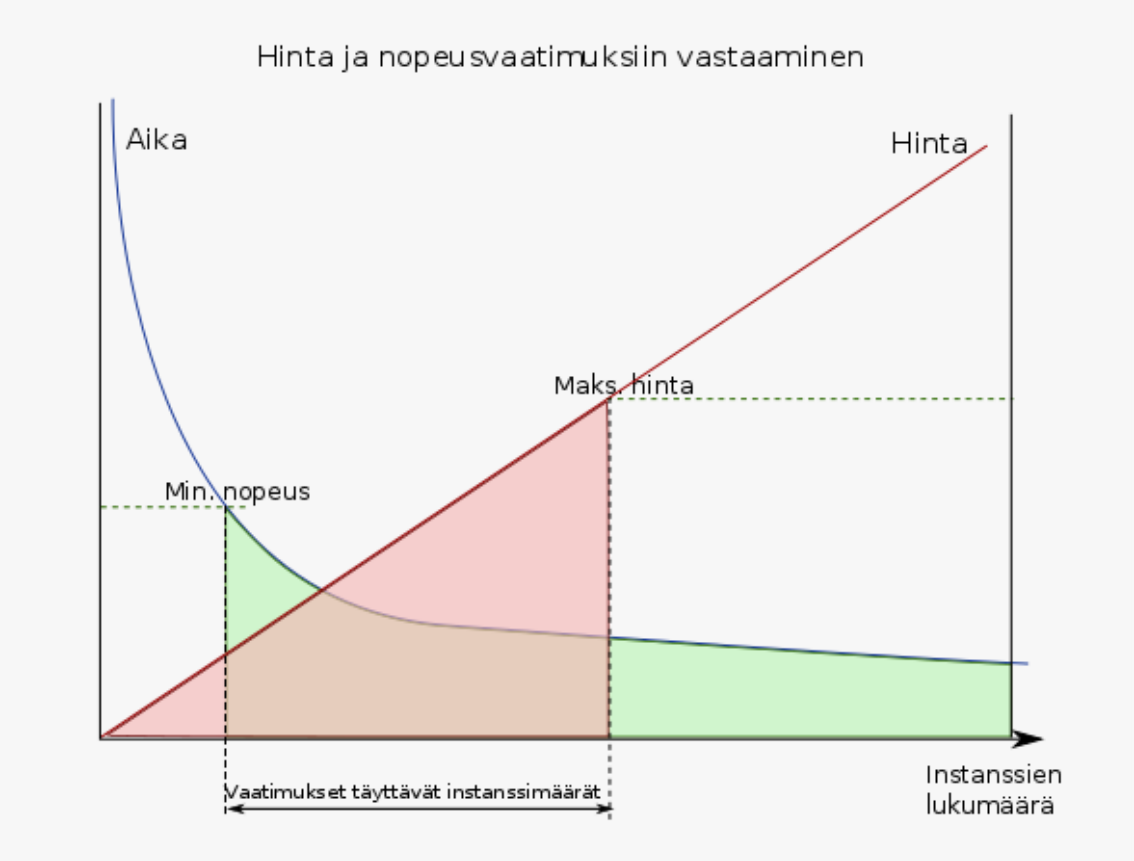
Kuva 7.2 *Minimilaskutushinta eli yhden tunnin hinta.*

Kuvassa 7.2 on esitetty yhden tunnin laskutushinnat eri instanssimäärillä. Nämä hinnat ovat samalla myös minimilaskutushintoja. Kasvaneeseen kuormaan ei tässä tapauksessa kannata varautua lyhyeksi aikaa, vaan on tarpeellista selvittää jollain tavalla

pidemmän aikavälin todennäköinen kuorma. Tämä voidaan toteuttaa esimerkiksi monitoroimalla järjestelmää ja selvittämällä ruuhkaisimmat ajankohdat. Myös työjonon pituudesta voidaan pyrkiä arvioimaan tarvittavan laskentatyön määrää, ja jos se ylittää tietyn rajan, käynnistetään uusia yksiköitä, joille riittää työtä vähintään tunniksi. Mikäli laskettavia videoita on jonossa riittävästi, ei instansseille tule hukka-aikaa, vaan laskutettava aika voidaan käyttää kokonaan hyödyksi.

7.2.2 Kannattavuus ja optimointi

Videomuunnokseen tai muutoinkin laskennan hajauttamiseen käytettävää laskentayksiköiden määrää valittaessa tulee ottaa huomioon laskennan ajallisen tehokkuuden lisäksi myös laskennasta koituvat kustannukset. Kuvassa 7.3 on esitetty edellä tehtyjen mittausten perusteella periaatteellinen kuva hajautettuun laskentaan käytetystä kokonaisajasta laskentayksiköiden määrän funktiona sekä vastaavasti hinnan kehityksestä eri laskentayksikkömäärillä. Erilaisissa sovelluskohteissa tai asiakaskunnissa voidaan asettaa laskennalle rajaehtoja, kuten yläraja kustannuksille tai minimivaatimus ajalliselle tehokkuudelle. Näitä on havainnollistettu kuvassa erivärisillä alueilla, jotka toteuttavat kunkin vaatimuksen erikseen. Jos molemmat ehdot halutaan samanaikaisesti toteuttaa, saadaan rajattua laskentayksiköiden määrä näiden rajojen väliselle alueelle.



Kuva 7.3 *Videomuunnokseen käytettävän optimaalisen instanssilukumäärän selvittäminen, kun hinnalle ja käytetylle laskenta-ajalle on asetettu rajoituksia.*

Aikakriittisissä sovelluksissa järjestelmän on voitava toteuttaa muunnos jossakin minimiajassa. Tämä voi riippua esimerkiksi kohdejärjestelmän halutusta käyttökokemuksesta tai joissain tapauksissa todellisista aikavaatimuksista, jolloin muunnoksen on valmistuttava ennen vaadittua aikarajaa. Usein vaadittava aikaraja ei ole ehdoton, vaan muunnos halutaan tapahtuvan riittävän nopeasti, jotta haluttu käyttökokemus voidaan saavuttaa. Esimerkiksi suoratoistopalveluun lähetettävän videon muuntamisessa ei ole hyvä kestää useita tunteja. Käyttäjä on kuitenkin valmis odottamaan jonkin aikaa videon valmistumista katselua varten. Tämän kaltaisia palveluita ovat esimerkiksi YouTube, Facebook ja muut sosiaalisen mediaan liittyvät videotoistopalvelut.

Esimerkki ehdottomasta aikavaatimuksesta on reaaliajassa tapahtuva muuntaminen, jota tarvitaan esimerkiksi television live-lähetysten toistamiseen. Tässä työssä toteutettua järjestelmää ei kuitenkaan voida käyttää tähän tarkoitukseen, koska eri laskentayksiköt muuntavat videosta eri ajankohdat. Reaaliajassa tapahtuvaa muuntamista varten videota ei voida jakaa eri yksiköille videon ajan perusteella, vaan tässä rajoitutaan muunlaisiin jakoperusteisiin. Yksi tämän kaltainen tapa olisi antaa eri kohdeformaatit eri instanssien tehtäväksi. Todennäköisesti tästä saavutettava hyöty ei olisi merkittävä, vaan muunnos kannattaisi tehdä yhdellä riittävän tehokkaalla instanssilla. Reaaliaikaista muuntamista varten on olemassa myös laitteita, joilla muunnos tapahtuu erityisesti sitä varten toteutetulla laitteistolla.

Järjestelmiä, jossa muunnokselle ei ole selkeää reaaliaikavaatimusta, voisivat olla esimerkiksi Netflix ja Yle Areena, joitain poikkeuksia lukuun ottamatta. Tällöin on tärkeää saavuttaa riittävä laskentakapasiteetti, mutta varsinaista aikavaatimusta ei ole, koska esitysajankohtaan voidaan varautua hyvissä ajoin. Tämän kaltaiseen järjestelmään tässä työssä kuvattu hajauttaminen olisi toteutettavissa. Tällöin voidaan myös suorittaa laskenta mahdollisimman vähillä laskentayksiköillä, ja pyrkiä minimoimaan aiheutuneet kustannukset.

8 JOHTOPÄÄTÖKSET JA YHTEENVETO

Videon koodin muuntaminen vaatii järjestelmästä hetkellisesti paljon resursseja. Itse muuntaminen on raskasta, mutta koodimuunnosjärjestelmän suorittamat videomuunnokset eivät yleensä jakaannu tasaisesti ajan suhteen. Vain suurimmilla järjestelmillä videonmuuntamistyötä on tarjolla niin paljon, että järjestelmä ei olisi satunnaisesti ilman tehtävää. Ja vaikka työtä riittäisikin, niin se kuormittaa järjestelmää yleensä epätasaisesti eri ajankohtina. Jos tätä laskentaa varten varataan erillinen tietokone, jonka suorituskyky on tasainen, saattaa ongelmaksi muodostua sen joutokäynti tai ylikuormitus. Tässä työssä yllä mainittuun ongelmaan pyrittiin saamaan ratkaisu käyttämällä palveluna tarjottavia alustoja, erityisesti julkisesti tarjolla olevia pilvipalveluja.

Pilvipalvelualustat tarjoavat erittäin joustavan ratkaisun sovelluksen vaakasuoralle skaalaukselle. Pilvipalveluna tarjottavat yksiköt voidaan käynnistää tarpeen mukaan lyhyeksikin ajaksi, ilman suurta alkuinvestointia. Pilvipalveluista maksetaan käytön mukaan, jolloin ne voidaan ottaa käyttöön silloin, kun niille on tarvetta. Hiljaisena aikana voidaan pitää ajossa pienempää määrää yksiköitä ja kiireisenä aikana vastaavasti kasvattaa määrää tehon lisäämiseksi. Näin voidaan välttää järjestelmien joutoajo, ja kuitenkin pystytään vastaamaan suureenkin kuormaan. Tavoitteena oli selvittää, miten suuri hyöty saavutettaisiin, kun yllä kuvatun kaltainen, hetkittäin vaihtuva kuormitus, lasketaan hajautetusti pilvessä. Tätä työtä varten toteutettu videokoodimuunnosjärjestelmä sekä sillä tehdyt mittaukset osoittavat, että hajauttaminen on mahdollista ja siitä voidaan saada merkittävää hyötyä vaihtuvaan kuormitukseen sopeuduttaessa.

Videotiedoston hajautettua laskentaa varten video joudutaan jakamaan jollain tavalla erillisiin osiin, jotka lasketaan toisistaan riippumatta. Diplomityötä varten toteutetussa demojärjestelmässä hajauttaminen toteutettiin jakamalla video lyhyisiin, ajallisesti peräkkäisiin osiin, jotka voidaan muuntaa itsenäisesti. Muunnettavat osat lähetettiin erillisille laskentayksiköille laskentaa varten ja lopuksi muunnetut osat liitettiin yhteen valmiiksi videoksi.

Demojärjestelmän avulla suoritettujen mittausten avulla havaittiin, että skaalauksesta saavutettava hyöty vähenee nopeasti laskennan hajautusta lisättäessä. Tämä on yksi merkittävimmistä hajauttamiseen liittyvistä haasteista. Kun videomuunnostyö hajautettiin eri laskentayksiköille, saavutettiin aluksi merkittävää tehonlisäystä. Laskettaessa viidellä yksiköllä yhden sijasta muunnostyön tehokkuus lisääntyi 55 prosenttia, mitä voidaan pitää merkittävänä nopeutuksena. Tämä trendi ei kuitenkaan jatkunut instanssimäärän kasvaessa. 25 laskentayksiköllä päästiin lähes yhtä hyvään tulokseen kuin 50 yksiköllä. Määrää edelleen kasvatettaessa hyöty jäi yhä pienemmäksi.

Edellä kuvattu tehokkuushyödyn väheneminen selittyy pääasiassa sillä, että verkkoliikenteen määrä kasvaa, kun hajautusastetta lisätään. Mitä useammalle yksikölle laskentaa hajautetaan, sitä enemmän tulee samanaikaista tiedonsiirtoa usealle erilliselle vastaanottajalle. Myöskään tässä työssä toteutettu tiedonsiirtotapa ei ole erityisen tehokas,

joten siinä olisi mahdollista optimoida järjestelmän toimintaa. Myös tiedonsiirtokapasiteetin lisääminen olisi perusteltua, jos hajautusta haluttaisiin lisätä. Tästä hankaluudesta huolimatta hajauttamisesta saavutettava hyöty on merkittävä, ja kokonaista järjestelmää voitaisiin skaalata vielä huomattavasti enemmän lisäämällä rinnakkain uusia tässä työssä kuvattuja järjestelmiä. Tämä edellyttää sitä, että laskettavana on useita erillisiä videoita, sillä yhden videon laskennan hajauttaminen ei tehostuisi tämän kaltaisella skaalaamisella.

Tässä työssä keskityttiin tarkastelemaan videomuunnoksen hajauttamista ajan suhteen osioidulla videolla. Jotta videokoodauksen hajautettu laskenta voitaisiin ottaa käyttöön tuotantoympäristössä, olisi mielestäni järkevää tutkia videon paloittelua tarkemmin. Voidaanko kaikenlaiset videoformaatit jakaa riittävän helposti osiin tai onko olemassa muita tapoja jakaa laskettava video itsenäisesti laskettaviksi kokonaisuuksiksi. Yksi mahdollinen käyttökohde, johon ei kuuluisi videon osiointi, olisi järjestelmä, jossa yksi muunnettava video joudutaan muuntamaan useaan eri kohdeformaattiin. Tällöin eri kohdeformaatin mukaan muunnettavat työt voitaisiin jakaa usealle erilliselle instanssille. Tällöin tosin joudutaan sama tiedosto välittämään samanlaisena usealle eri kohdetietokoneelle.

Toinen mahdollinen jatkotutkimuskohde liittyy reaaliaikaiseen videon muuntamiseen. Tässä työssä toteutettu järjestelmä ei sellaisenaan sovellu videon reaaliaikaiseen muuntamiseen. Reaaliaikaisesti muunnettavaa videota ei voida hajauttaa jakamalla se ajallisesti peräkkäisiin, tietyn mittaisiin osiin, vaan silloin joudutaan muuntamaan aina lähes nykyhetkeä vastaava videon kohta. Reaaliaikaisen muunnoksen hajauttamista varten olisi löydettävä jokin tapa hajauttaa video muulla tavalla osiin. Eräs mahdollinen tapa jakaa video reaaliaikaista laskentaa varten osiin, on jakaa se kuva-alan perusteella. Tällöin esimerkiksi neljään osaan jaetun videon transkoodaus voitaisiin hajauttaa siten, että yksi tietokone saa laskettavakseen vasemman ylänurkan, toinen oikean ylänurkan ja niin edelleen. Tämä on kuitenkin teknisesti haastavampaa kuin videon jakaminen ajan perusteella.

Diplomityössä saavutettiin asetetut tavoitteet toteuttamalla pilvipohjainen, hajautettu videon transkoodausjärjestelmä laskennan hajauttamiseen liittyvän teorian pohjalta. Tämän työn perusteella voidaan havaita, että laskennan hajauttaminen on järkevästi toteutettavissa pilvipalveluita hyväksi käyttäen myös videon muuntamiseen.

LÄHTEET

[1] Responsibilities in the Cloud. Kavis Technology Consulting. [WWW]. [Viitattu 22.4.2016]. Saatavilla: http://www.kavistechnology.com/blog/responsibilities-in-the-cloud/

[2] Melvin B. Greer. 2011. Software As a Service Inflection Point: Using Cloud Computing to Achieve Business Agility, iUniverse 180 s.

[3] Armbrust, M. et al. 2009. Above the clouds: a Berkeley view of cloud computing. Technical report, UCB/EECS-2009-28, University of California, Berkeley. 22 s.

[4] X. Liu et al. 2012, The Design of Cloud Workflow Systems, SpringerBriefs in Computer Science, The Author(s). 89 s.

[5] Rountree, Derrick, Ileana Castrillo. 2014. The Basics of Cloud Computing. Syngress Publishing. 155 s.

[6] Schlossnagle, Theo. 2007. Scalable internet architecture. Indianapolis, Ind.: Developer's Library Sams, 264 s.

[7] Hanzo, Lajos et al. 2007. Video compression and communications: from basics to H.261, H.263, H.264, MPEG2, MPEG4 for DVB and HSDPA-style adaptive turbo-transceivers. IEEE Press. 635 s.

[8] Thyagarajan, K. Basics of Video Compression. 2011. Primo Central Index (Ex Libris) - John Wiley & Sons, Inc. 421 s.

[9] Takamura, Seishi, and Yoshiyuki Yashima. 2005. "H. 264-based lossless video coding using adaptive transforms." *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*. Vol. 2.

[10] Adalia Oy sisäinen aineisto. 2013. Adalia Foundation.Video API.

[11] Vimeo. 2014. Video Compression Guidelines. [WWW]. [Viitattu 16.4.2014]. Saatavissa: http://vimeo.com/help/compression

[12] Brightcave. 2014. Video encoding best practices. [WWW]. [Viitattu 16.4.2014]. Saatavissa: http://support.brightcove.com/en/video-cloud/solutions/video-encoding-best-practices

[13] Streaming vs. Progressive Download. Streaming Media Magazine (Jun/Jul 2007), s. 54-58.

[14] Xin, Jun, Anthony Vetro, and Huifang Sun. 2004. Efficient macroblock coding-mode decision for H. 264/AVC video coding. Picture coding symposium.

[15] Ostermann, Jörn, et al. 2004. Video coding with H. 264/AVC: tools, performance, and complexity. Circuits and Systems magazine, IEEE 4.1. s. 7-28.

[16] Mediatrade Oy. 2014. Videosignaalin transkoodaus ja uudelleen enkoodaus. [WWW]. [Viitattu: 17.4.2014] Saatavilla: http://www.mediatrade.fi/ajankohtaista/2010/02/24/videosignaalin-transkoodaus-ja-uudelleen-enkoodaus/

[17] MPlayer - The Movie Player Dokumentaatio. [WWW]. [Viitattu 17.4.2014]. Saatavilla: http://www.mplayerhq.hu/DOCS/HTML/en/menc-feat-x264.html#menc-feat-x264-encoding-options-misc-preferences

[18] Wowza Streaming Engine™ käyttöopas. [WWW]. [Viitattu 17.4.2014]. Saatavilla: http://www.wowza.com/resources/WowzaStreamingEngine_UsersGuide.pdf

[19] Jan Ozer, What is HLS (HTTP Live Streaming)? Kirjoitettu 14.10.2011 [WWW]. [Viitattu 17.4.2014]. Saatavilla: http://www.streamingmedia.com/Articles/Editorial/What-Is-.../What-is-HLS-%28HTTP-Live-Streaming%29-78221.aspx

[20] Microsoft. Smooth Streaming. [WWW]. [Viitattu 17.4.2014]. Saatavilla: http://www.iis.net/downloads/microsoft/smooth-streaming

[21] ØMQ Messaging Patterns, Client/Server. [WWW]. [Viitattu 12.4.2016]. Saatavilla: http://learning-0mq-with-pyzmq.readthedocs.io/en/latest/pyzmq/patterns/client_server.html

[22] Amazon earnings numbers show AWS success, IaaS growth. [WWW]. [Viitattu 24.2.2016]. Saatavilla: http://searchaws.techtarget.com/opinion/Amazon-earnings-numbers-show-AWS-success-IaaS-growth

[23] Cloud Computing and Sustainability: The Environmental Benefits of Moving to the Cloud. [WWW]. [Viitattu 24.4.2016]. Saatavilla: http://gesi.org/files/Reports/AssessmentMethodologyCasteStudy_CloudComputingSustainability-Nov2010.pdf

[24] Big Buck Bunny. [WWW]. [Viitattu 24.4.2016]. Saatavilla: https://peach.blender.org/media-gallery/

[25] Amazon EC2 Pricing. [WWW]. [Viitattu 28.4.2016]. Saatavilla: https://aws.amazon.com/ec2/pricing/

[26] Adaptive Bitrate Streaming, Wikipedia. Kuvan tekijä Wikipedia-käyttäjä Dasedon. [WWW]. [Viitattu 8.5.2016]. Saatavilla: https://en.wikipedia.org/wiki/Adaptive_bitrate_streaming

[27] ZeroMQ. [WWW]. [Viitattu 8.5.2016]. Saatavilla: http://zeromq.org/

[28] Great Internet Mersenne Prime Search GIMPS. [WWW]. [Viitattu 9.5.2016]. Saatavilla: http://www.mersenne.org/

[29] Choi, J. 1998. A new parallel matrix multiplication algorithm on distributed-memory concurrent computers. Concurrency-Practice And Experience, 1998 Jul, Vol.10(8), s. 655-670