SATAKUNNAN AMMATTIKORKEAKOULU

Julius Kärkiluoma

HAJAUTETTU VIDEONPAKKAUSOHJELMISTO

Tietotekniikan koulutusohjelma

Ohjelmistotekniikka

2006

HAJAUTETTU VIDEONPAKKAUSOHJELMISTO

Kärkiluoma, Julius
Satakunnan ammattikorkeakoulu
Tekniikka Pori
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka
Joulukuu 2006
Aarinen, Reino
UDK: 004.41 ; 004.42 ; 004.43
Sivumäärä: 39

Avainsanat: multimedia, Python, video, hajautus

Opinnäytetyössä tutkittiin videon pakkaamiseen kuluvan ajan lyhentämistä. Työhön kuului hajautetun videonpakkausohjelman ohjelmoiminen. Ohjelman tarkoituksena on lyhentää videonpakkaukseen kuluvaa aikaa jakamalla pakkauksesta aiheutuva kuorma monen koneen kesken. Ohjelma suunniteltiin niin, että sitä voi käyttää jo kahden koneen avulla. Hajautettu videonpakkausohjelmisto ohjelmoitiin Python ohjelmointikielellä.

Ohjelma käyttää hyväkseen vapaasti ladattavaa x264 ohjelmaa, joka pakkaa videon käyttäen H.264/MPEG-4 AVC videonpakkausta. H.264/MPEG-4 AVC valittiin videonpakkausformaatiksi koska se on yksi uusimmista koodekeista ja täten myös yksi tehokkaimmista. Pakattavaksi videotiedostoksi valittiin YUV4MPEG2 formaatissa oleva tiedosto. Palvelimelle ja asiakaskoneille ohjelmoitiin omat ohjelmat. Ohjelman tekemiseen käytettiin ainoastaan ilmaisia työkaluja.

Ohjelman toteuttamisessa onnistuttiin tavoitteiden mukaisesti. Ohjelman jatkokehittely suunnitelmia mietittiin ja tultiin siihen tulokseen, että ohjelmaa on mahdollista vielä kehittää eteenpäin, jolloin siitä saattaa tulla sopivalle alan yritykselle tai palvelijalle sovitettuna jopa menestyksekäs videonpakkausohjelma.

DISTRIBUTED VIDEO ENCODING SOFTWARE

Kärkiluoma, Julius
Satakunta University of Applied Sciences
School of Technology Pori
Degree Programme in Infomation Technology
Software Engineering
December 2006
Aarinen, Reino
UDK: 004.41 ; 004.42 ; 004.43
Number of pages: 39

Key words: multimedia, Python, distributed video encoding

The aim for this study was to program distributed video encoding software. The distributed video encoding software would shorten the time needed to encode video by distributing the load across several computers. The requirements for this application were designed so that it is usable using two or more computers. The software was programmed with the Python programming language.

The software uses the free x264 application, which encodes video using the H.264/MPEG-4 AVC video codec. This codec was chosen because it uses the latest video encoding methods and therefore causes a heavy load on processors, which was ideal for the distributed video encoding software. The software accepts video in the YUV4MPEG2 format as an input file. The server and the client(s) use different codes. Only free tools and techniques were used in the programming of the code.

Programming the application as planned succeeded. Future improvements were researched and it was found that the application has potential to grow into a successful video encoding application.

SISÄLLYS

1. JOHDANTO....5
2. MPEG-4 AVC....6
2.1 Värikylläisyyden aliotanta....7
2.2 Liikevektorit....7
2.3 Kehystyypit (I,P,B,SP,SI)....9
2.4 Vaihteleva bittinopeus....9
2.5 Multi-pass pakkaus....10
2.6 Algoritmit....10
3. HAJAUTETTU VIDEONPAKKAUSOHJELMISTO....10
3.1 Olemassa olevia ratkaisuja....10
3.2 Ohjelman kuvaus....11
3.3 Käytetyt tekniikat....11
3.3.1 Python....11
3.3.2 x264....12
3.3.3 YUV4MPEG2....12
3.4 Ohjelman toiminta....13
3.4.1 Palvelin....13
3.4.2 Asiakas....15
4. TESTAUS....16
5. YHTEENVETO....22
LÄHTEET....25
LIITTEET

SYMBOLI JA TERMILUETTELO

DCT	DCT (Discrete Cosine Transform) eli diskreetti kosinimuunnos on häviöllinen algoritmi, jonka avulla kuvasta voidaan karsia ylimääräistä informaatiota. Videokuvaa pakattaessa käytetään kaksiulotteista muunnosta, joissa muuttujina ovat korkeus- ja leveyskoordinaatit. Kullekin makrokehykselle lasketaan DCT-kertoimet ja kuvalohko esitetään näistä kertoimista koostuvana matriisina. Koska ihmissilmä ei erota tarkasti pieniä värin muutoksia, voidaan eri kuvalohkoja koodata samalla väriarvolla ja korvata kertoimia keskiarvoilla. Menetelmää käytetään lähes kaikissa häviöllisissä kuvanpakkausmenetelmissä. /11/
FPS	FPS (Frames Per Second) kuvaa kuinka monta kehystä sekunnissa videokuvassa näytetään.
Häviöllinen pakkaus	Häviöllisessä (lossy) pakkauksessa videokuvan pakkaamisen jälkeen videokuvaa ei pystytä purkamaan entiselleen. Pakkauksen yhteydessä tietoa ”hävitetään”, jotta tiedostokoko saataisiin pienemmäksi.
Häviötön pakkaus	Häviöttömässä (lossless) pakkauksessa videokuvan purkamisella päästään alkuperäiseen tilanteeseen, jossa videokuva oli ennen pakkaamista. Häviöttömässä pakkauksessa tieto säilyy ennallaan. Häviöttömän pakkauksen videota saatetaan kutsua myös ”pakkaamattomaksi”.
Kehys	Videokuva koostuu kehyksistä (kuvista), joita näytetään tietyllä nopeudella, jolloin videokuvassa tapahtuva liike saadaan aikaan. Kehyksien nopeutena voidaan käyttää esim. 25 kehystä sekunnissa.
Koodekki	Koodekin (codec) tehtävänä on videotiedoston pakkaus (encoding) ja purku (decoding).
Makrokehys	Videotiedosto jaetaan aina pienempiin osiin, joissa tieto käsitellään. Esim. 320*240 pikselin videotiedosto saattaa olla jaettuna 80*60 makrokehykseen, joista jokainen sisältää 4*4 pikseliä.
Värikylläisyys	Värikylläisyydellä tarkoitetaan värin eroavaisuutta harmaasta, kun värin kirkkaus sekä värisävy on väristä määritelty.

1. JOHDANTO

Videon pakkaus on menetelmä, jossa videotiedoston kokoa pyritään pienentämään. Videota pakataan mm. poistamalla siitä asioita, joihin ihmissilmä ei kiinnitä huomiota. Ideaalitilanteessa videon visuaalisen laadun muutosta ei pystytä havaitsemaan, vaikka tiedoston koko onkin pakkautunut huomattavasti pienempään tilaan.

Tekniikan kehittyessä tarve suuriresoluutioisille videotiedostoille kasvaa. Tämä edellyttää parempien pakkaustekniikoiden kehittämistä. Paremmat pakkaustekniikat tarvitsevat myös enemmän prosessoritehoa. Samalla kun tietokoneet kehittyvät, myös uusien pakkaustekniikoiden laitevaatimukset ovat nousseet. Uusimpien koodekkien käyttöön kotikoneilla vaaditaan markkinoiden tehokkaimpia koneita.

Vaikka tietokoneiden massamuistit kasvavat kokoajan suuremmiksi, on tarvetta pienentää videotiedostoja pakkaamalla niitä. Pakkaamattomiakin (häviöttömiä) videotiedostoja käytetään yleisesti, lähinnä kun videokuvaa editoidaan ja sitä pitää pakata uudelleen monta kertaa.

Videonpakkauksen muuttuessa yhä raskaammaksi on tarve kehittää erilaisia ratkaisuja pakkauksen nopeuttamiseksi. Hajautetun videonpakkausohjelmiston tarkoituksena on jakaa videonpakkauksesta aiheutuva kuorma monen tietokoneen kesken, jolloin pakkaukseen tarvittava aika lyhenee.

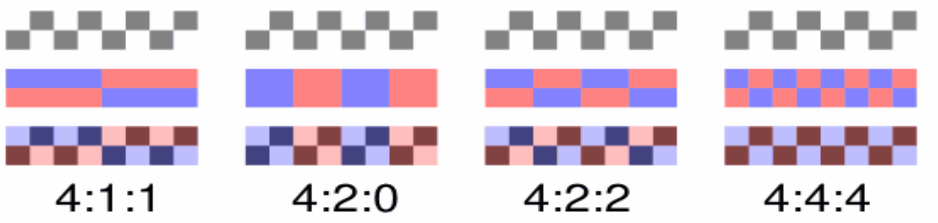
2. MPEG-4 AVC

MPEG-4 standardi (ISO-14496) sisältää monia eri osa-alueita, jotka käsittelevät erilaisia multimedian esitystapoja digitaalisessa mediassa. Tässä luvussa esitellään ITU-T:n Video Coding Experts Group (VCEG) ja ISO/IEC:n Moving Picture Experts Group (MPEG) organisaatioiden yhteistyönä suunnittelemaa H.264/MPEG-4 AVC videonkoodausta, joka kuuluu MPEG-4:n osaan 10 (Part 10). Nimitys H.264 tulee ITU-T:n H.26x koodausmenetelmistä ja AVC (Advanced Video Coding) tulee ISO/IEC organisaation puolelta. MPEG-4 AVC mahdollistaa häviöllisen pakkaustavan lisäksi myös häviöttömän pakkauksen.

Videonpakkauksessa kaikkein huonoin ratkaisu on suoraan kuvassa olevan informaation kertominen. Videota pystytään pakkaamaan pienempään kokoon kertomalla epäsuorasti videon tapahtumista ja mm. ihmissilmälle huomaamattomia asioita yritetään poistaa videokuvasta. Seuraavaksi kuvataan erilaisia videonpakkausmenetelmiä, joita H.264/MPEG-4 AVC:llä häviöllisesti pakattu video käyttää hyväkseen.

2.1 Värikylläisyyden aliotanta

Välikylläisyyden aliotannassa (Chroma subsampling) on kyse siitä, että luminanssiin panostetaan enemmän kuin pelkkien värien kerrontaan. Tämä tehdään sen vuoksi, että ihmissilmä pystyy erottamaan tarkemmin kuvassa tapahtuvat kirkkauden muutokset kuin värien muutokset. Värikylläisyyskomponentit on mahdollista muodostaa laskukaavojen avulla RGB (Red Green Blue)-komponenteista. Muutoksen jälkeen väritekniikan kuvausta kutsutaan nimellä YCbCr. Y kuvaa luminanssia (kirkkaus), Cb kuvaa sinisen värin värikylläisyyttä ja Cr kuvaa punaisen värin värikylläisyyttä. Värikylläisyyden aliotannassa on käytössä eri suhteita kuvaamaan komponenttien osuuksia. Esim. 4:4:4 suhteella jokaista komponenttia on saman verran (Kuva 1). /1, 2 s. 15/



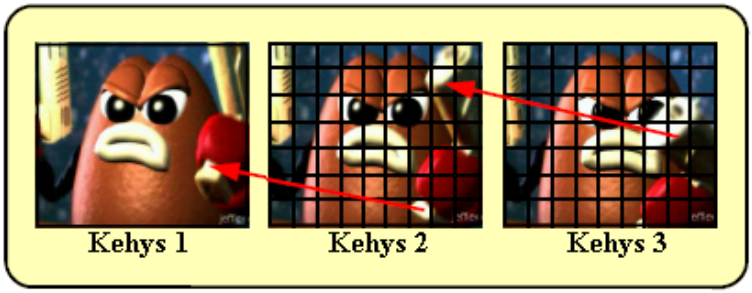
Kuva1. Värikylläisyyden aliotannassa käytettävät eri YCbCr suhteet. /1/

2.2 Liikevektorit

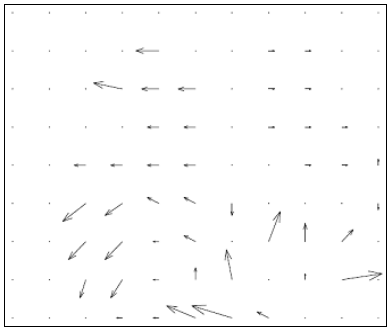
Liikevektoreilla kerrotaan kuvassa tapahtuva jokaisen makrokehyksen liike ja liikkeen suunta (Kuva 2). Yksi liikevektori kuvaa aina yhden makrokehyksen liikkumista (Kuva 3).

Pakattavan videon jokainen kehys jaetaan aina pienempiin osiin, joissa videon kuvainformaatio käsitellään. Kehyksen osia kutsutaan makrokehyksiksi. Makrokehysten koko saattaa olla esim. 4*4 pikseliä. H.264/MPEG-4 AVC tukee toiminnoissaan vaihtelevaa

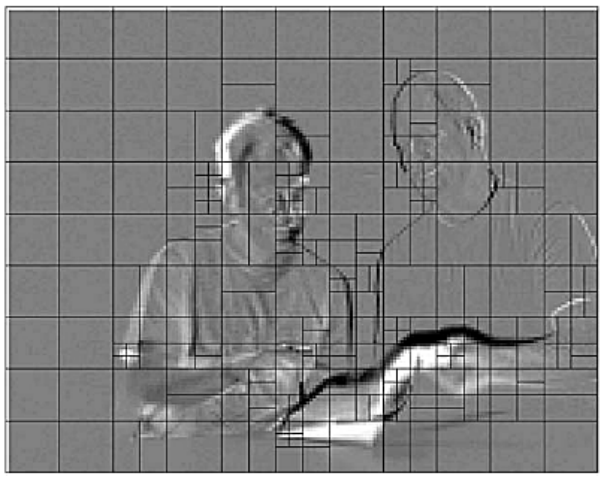
makrokehyskokoa, mikä mahdollistaa vaativimpien kohteiden kuvaamisen pienemmillä makrokehyksillä (Kuva 4). Pienempi makrokehyskoko tarkoittaa sitä, että kehyksen pikseleitä voidaan kuvata tarkemmin.



Kuva 2. Edelliseltä kehykseltä etsitään samanlaisia makrokehyksiä. Samanlaisen makrokehyksen löytyessä makrokehys voidaan kuvata liikevektorin avulla. /3/



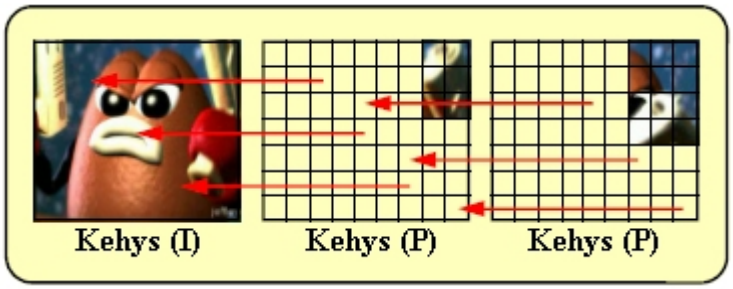
Kuva 3. Liikevektorien suunta ja nopeus (pituus) 16*16 makrokehyksen kuvassa. /2 s. 40/



Kuva 4. Makrokehykset jaetaan pienempiin osiin, jos makrokehyksissä on tietoa, jonka kuvaamiseen tarvitaan enemmän bittejä. /2 s. 172/

2.3 Kehystyypit (I,P,B,SP,SI)

I-kehys on perustana kuvalle sisältäen eniten alkuperäistä kuvainformaatiota. P-kehyksiä sijoitetaan I kehyksien väliin. P-kehyksellä viitataan aina johonkin I tai P kehykseen ja ilmaistaan ainoastaan liikkuvassa kohteessa tapahtunut muutos (Kuva 5). B-kehykset sijaitsevat I ja P kehysten välissä. B-kehykset täydentävät P-kehysten informaatiota. SP ja SI kehykset ovat tarkoitettu tilanteeseen, jossa vaihdetaan koodivirtaa (coded stream). Aikaisemmissa koodekeissa kehykset ovat voineet ainoastaan viitata yhteen tai kahteen eri kehykseen. H.264 mahdollistaa kehyksien viittaukset kehyslistasta, joka mahdollistaa jopa useiden kymmenien kehyksien käyttämisen viitteinä. /2 s.164, 4 s.220/



Kuva 5. Videossa tapahtuvat muutokset ainoastaan kerrotaan. Muuttumaton osa kuvasta pysyy kuvassa samana, jolloin sitä on turha esittää uudestaan. /3/

2.4 Vaihteleva bittinopeus

Vaihtelevalla bittinopeudella (VBR / Variable Bit Rate) pyritään optimoimaan videokuvan laatua eri kohdissa teosta siten, että nopean toiminnan kohtauksille uhrataan enemmän tallennustilaa kuin stabiileille otoksille /9/. Itse videotiedoston kokoa näin ei pystytä pienentämään, mutta videokuvan laatu paranee jonkin verran. Tasaisella bittinopeudella (CBR / Constant Bit Rate) jokaista kehystä kohden on käytettävissä yhtä monta bittiä, jolloin kehyksien koot pysyvät samana kokoajan. Tasaisella bittinopeudella on helpompi ennustaa pakkauksen jälkeistä tiedostokokoa. Vaihtelevaa bittinopeutta käytettäessä video pitää yleensä analysoida ensiksi kertaalleen, ja sen jälkeen pakata video, jos halutaan määrittää etukäteen pakatun videon tiedostokoko.

2.5 Multi-pass pakkaus

Multi-pass pakkaus tarkoittaa sitä, että ennen videon pakkausta koodekki analysoi tiedoston. Tämän analysoinnin avulla selviää, mihin kannattaa bittejä uhrata ja mihin ei. Tekniikka toimii vain VBR tiedostoja tehtäessä. Analysoitaessa tehdään lokitiedostoa, johon tallennetaan tiedot kohtien monimutkaisuudesta pakkauksen suhteen. Pakatessa koodekki käyttää tätä lokitiedostoa avukseen ja valitsee sopivat bittinopeudet tarpeen mukaan. /5/

2.6 Algoritmit

Algoritmit ovat tärkeä osa videonpakkauksen tiedontiivistämisessä. MPEG-4 AVC käyttää monia eri algoritmeja häviöttömään ja häviölliseen pakkaukseen. Aritmeettisessa koodauksessa (CABAC) tieto käsitellään symboleina ja niiden esiintymisille lasketaan todennäköisyyksiä, joiden perusteella tieto rakennetaan uudelleen. Muunnoskoodauksessa (Transform Coding) on kyse lukusarjojen muuntamisesta muotoon, jossa osa lukujen merkityksestä vähenee, jolloin lukujen määrää voidaan vähentää. MPEG-4 AVC käyttää yleisen DCT muunnoskoodauksen tapaista tehokkaampaa koodausta. /2 s.165 /

3. HAJAUTETTU VIDEONPAKKAUSOHJELMISTO

3.1 Olemassa olevia ratkaisuja

Hajautettuja videonpakkausohjelmistoja ei ole monia saatavilla ja ohjelmat ovat vasta varhaisessa kehitysvaiheessa.

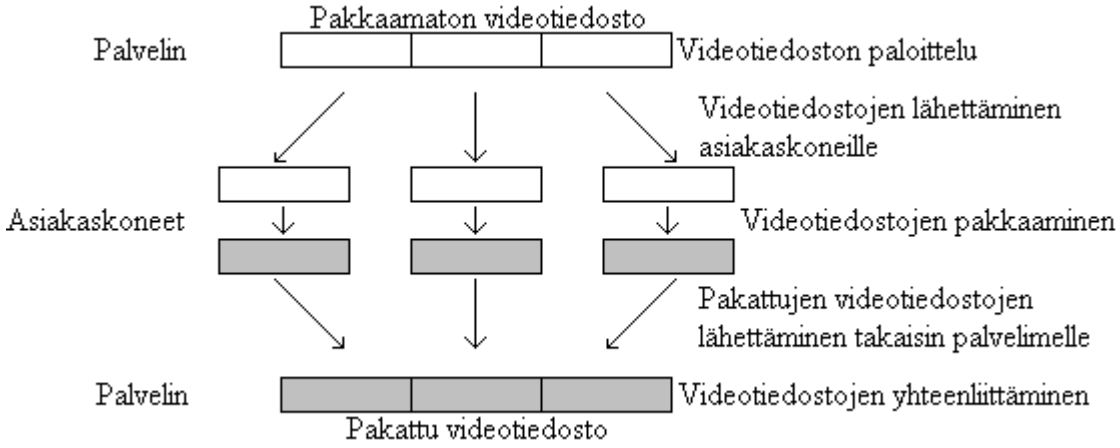
GridIron Softwaren (http://www.gridironsoftware.com) tuotteet on kehitetty videonkäsittelyn ammattilaisille. Yrityksen nykyiset tuotteet käyttävät epäsuorasti hajautettua videonpakkausta hajauttamalla laskentakuormaa moniydinsuorittimille. Yrityksellä on ollut myös suunnitelmia tuoda peruskäyttäjien ulottuville hajautettuun videonkäsittelyyn perustuvia ratkaisuja /10/.

Hajautettu videonpakkausohjelmisto ELDER (http://www.funknmary.de/bergdichter/pro

jekte/index.php?page=ELDER) käyttää AviSynth kehyspalvelinta sekä x264 ja XVID koodekkeja hyväkseen videonpakkauksessa. Vaikka ohjelmaa on kehitetty jo yli vuosi, se on vielä beta vaiheessa.

3.2 Ohjelman kuvaus

Hajautetulla tietojenkäsittelyllä pyritään laskentakuorman jakamiseen eri koneiden välillä. Hajautetun videonpakkausohjelmiston tarkoituksena on nopeuttaa aikaa vievää videonpakkaamista jakamalla videotiedosto monen eri koneen kesken ja näin ollen lyhentää videonpakkaukseen tarvittavaa aikaa (Kuva 6). Ohjelma on tarkoitettu 2-20 koneen lähiverkkoympäristöihin. Koska videotiedostot saattavat olla hyvinkin suuria, niin ohjelma ei sovellu hyvin internetin kautta käytettäväksi. Tiedostojen salausta ohjelmassa ei ole, koska se on suunniteltu lähinnä suljettuihin lähiverkkoympäristöihin. Tarvittaessa salauksen toteuttaminen ohjelmaan onnistuu helposti käyttämällä Pythonin valmiita kirjastoja.



Kuva 6. Ohjelman toimintaperiaate.

3.3 Käytetyt tekniikat

3.3.1 Python

Python valittiin ohjelmointikieleksi koska se on monipuolinen tulkattava ohjelmointikieli, joka on alun perin kehitetty yhdistämään skriptikielten ja tavanomaisten ohjelmointikielten hyvät puolet. Pythonin syntaksi on selkeää ja siihen on saatavilla moni-

puolisia kirjastoja eri käyttötarkoituksia varten. Python-tulkki ja -kirjastot on kehitetty avoimen lähdekoodin projekteina ja niitä levitetään Pythonin oman lisenssin (Python Software Foundation License) alaisena, joka on yhteensopiva myös GPL-lisenssin kanssa. Python tukee monenlaisia lähestymistapoja ohjelmointiin; sitä voi käyttää mm. oliopohjaisena, rakenteellisena tai funktionaalisena ohjelmointikielenä. Python on myös alustariippumaton. /3/

3.3.2 x264

Hajautettu videonpakkausohjelmisto käyttää x264.exe:ä videonpakkaamiseen. x264 on ilmainen GPL-lisenssin alainen H.264/MPEG-4 AVC yhteensopivaa videonpakkausta käyttävä ohjelma. Videonpakkausohjelman tekijöiden käsialaa on myös mm. suosittu VLC mediasoitinohjelma. x264 valittiin hajautettuun videonpakkausohjelmistoon, koska se käyttää H.264/MPEG-4 AVC yhteensopivaa videonpakkausta, joka on laadultaan yksi parhaimpia pakkausmenetelmiä /4/. H.264/MPEG-4 AVC käyttää hyväkseen uusimpia pakkausmenetelmiä, joten se on myös erittäin vaativa laskentatehon suhteen. Videonpakkausohjelman kotisivut löytyvät osoitteesta http://developers.videolan.org/x264.html ja päivittäiset käännetyt versiot on saatavilla osoitteesta http://x264.nl/. x264 ohjelmalla pakattava tiedosto pitää olla .y4m muodossa tai siinä pitää olla tuki AviSynth kehyspalvelimelle. Ohjelmalla voi pakata tiedostoja .264 (raaka bittivirta), .mkv (Matroska kääre) tai .mp4 muotoon.

3.3.3 YUV4MPEG2

Pakattavan kohdetiedoston koodekiksi valittiin häviötöntä pakkausta käyttävä koodekki, koska häviöllisen tiedoston uudelleenpakkaaminen ei ole suositeltavaa kuvanlaadun huononemisen takia. x264 ohjelmassa on tuki .y4m tiedostoille, mikä vaikutti osaltaan myös YUV4MPEG2:n valintaan. YUV4MPEG2 tiedoston kehykset ovat YCbCr 4:2:0 muodossa. YUV4MPEG2 tiedoston alkutunniste alkaa aina yhdeksäntavuisella ”YUV4MPEG2” tunnuksella, jota seuraa tyhjä väli. Alkutunnisteessa on myös kerrottu videon ominaisuuksista. Jokainen ominaisuus erotetaan toisistaan välilyönnillä alkutunnisteessa: kehyksen leveys (W, esim. W320), korkeus (H), FPS (F), kuvan lomitus (I) ja kuvasuhde (A). Alkutunnisteen jälkeen seuraa määrittämätön määrä kehyksiä, jotka alkavat kuuden tavun ”FRAME” merkinnällä, jota seuraa 0A-heksadesimaali.

YUV4MPEG2 tiedostokoko tavuina voidaan laskea seuraavalla tavalla: kehyksen korkeus * kehyksen leveys * 3/2 * kehyksien määrä. /5/

3.4 Ohjelman toiminta

3.4.1 Palvelin

Kun palvelinohjelma käynnistetään, ohjelma tarkistaa onko palvelimella x264.exe tiedostoa. Jos tiedostoa ei löydy ohjelma sulkeutuu. x264.exe tiedoston löytyessä ohjelma tulostaa alkuvalikon. Alkuvalikosta on mahdollisuus muuttaa palvelimen asetuksia valitsemalla A.

Asetuksista voidaan asettaa tallennettavan pakatun tiedoston nimi, sekä itse pakattavan tiedoston nimi. Pakattava tiedosto tulee olla .y4m videotiedosto ja tallennettavalle pakatulle tiedostolle on suositeltavaa käyttää .264 tiedostopäätettä. Ohjelma antaa virheilmoituksen jos se ei löydä pakattavaa tiedostoa. Tiedonsiirtoa varten asetuksista määritetään käytettävä tiedonsiirtoportti. Asetuksista voidaan määrittää osallistuuko itse palvelin myös pakkaukseen. Jos palvelin osallistuu pakkaukseen, niin käyttäjällä ei tarvitse olla kuin yksi asiakaskone. Palvelimen asetuksista voidaan määrittää asiakaskoneiden määrä. Tämän perusteella itse pakattava videotiedosto jaetaan osiin. Asiakaskoneiden määrää ei voida alentaa yhteen, jos palvelin ei osallistu pakkaukseen. Asetuksista voidaan määrittää myös poistetaanko lopuksi ohjelman väliaikaistiedostot (leikatut videot).

Asetuksien laiton jälkeen takaisin alkuvalikkoon pääsee kirjoittamalla T ja palvelinohjelmisto käynnistetään V kirjaimella. Ensimmäiseksi ohjelma tarkastaa pakattavan .y4m tiedoston oikeellisuuden etsimällä tiedoston alkutunnisteesta (header) merkkijonon ”YUV4MPEG2”. Ohjelma lopetetaan, jos kyseistä merkkijonoa ei löydy.

Seuraavaksi ohjelma etsii videotiedoston kaikki kehykset ja kehyksien sijainnit tallennetaan muistiin. Kehykset löydetään YUV4MPEG2 tiedostosta etsimällä ”FRAME” merkkijonoja. Koska videotiedosto saattaa olla erittäin suurikokoinen, jolloin kokonaisen tiedoston tallentaminen keskusmuistiin olisi mahdotonta, tiedostonkäsittelyssä käytetään puskuria (buffer). Puskuriin tallennetaan kerrallaan aina pieni määrä tietoa. Kun tieto on käsitelty puskurista, puskuri tyhjennetään ja sinne siirretään puskurin koon verran uutta

käsiteltävää tietoa lisää.

Seuraavana on vuorossa videotiedoston paloittelu. Ohjelma tallentaa videon alkutunnisteen muistiin (kaikki tieto ennen ensimmäisen kehyksen alkua). Video leikataan sen perusteella kuinka monta asiakasta palvelimelle on asetuksissa merkitty huomioiden myös palvelimen oman osuuden. Palvelin etsii kehyskohdat, joista video leikataan ja leikatut videotiedostot tallennetaan erillisiksi videotiedostoiksi. Jokaiseen leikattuun videotiedostoon tallennetaan myös videon alkutunniste alkuperäisestä videosta.

Videoiden tallentamisen jälkeen palvelin käynnistää uuden säikeen ohjelmassa ja alkaa pakata ensimmäistä leikattua tiedostoa, jos asetuksissa on määrätty, että palvelin osallistuu pakkaukseen. Jos palvelin ei osallistu pakkaukseen, uutta säiettä ei käynnistetä. Ohjelman säikeistäminen mahdollistaa usean samanaikaisen toiminnon suorittamisen. Palvelin pystyy pakkaamaan oman osuutensa videosta samalla kun palvelin lähettää leikattuja videotiedostoja asiakaskoneille.

Seuraavaksi palvelin jää odottamaan asiakaskoneiden yhteydenottoa. Jokainen uusi palvelin-asiakas yhteys käynnistetään uudelle säikeelle, mikä mahdollistaa palvelimen yhtäaikaisen tiedonsiirron monen asiakaskoneen välillä. Palvelin lähettää jokaiselle asiakaskoneelle yhden leikatun videotiedoston, minkä jälkeen palvelin jää odottamaan kunnes asiakaskoneet ovat pakanneet tiedostot. Palvelimen ja asiakaskoneen yhteydenpidosta on kerrottu tarkemmin asiakaskoneen toimintaa käsittelevässä luvussa.

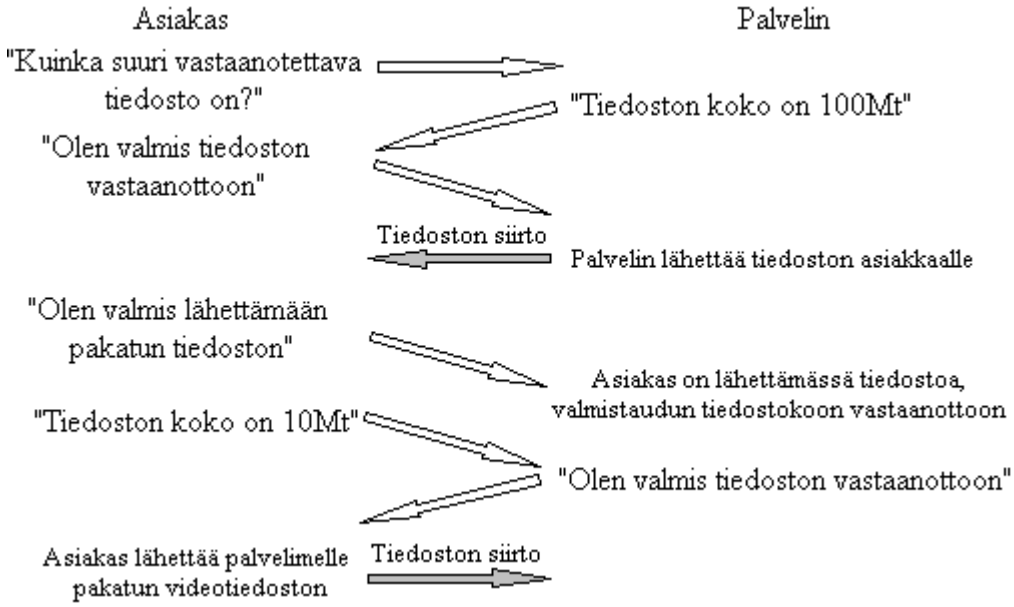
Palvelin sulkee yhteyksiä asiakaskoneisiin sitä mukaa kun asiakkaat ovat lähettäneet pakatut videotiedostot takaisin palvelimelle. Tämän jälkeen palvelin yhdistää tiedostot yhdeksi videotiedostoksi. Ennen ohjelman lopetusta väliaikaistiedostot poistetaan, jos siten on asetuksissa määritelty. Näin prosessi on saatu päätökseen.

3.4.2 Asiakas

Kun asiakasohjelma käynnistetään, ohjelma tarkistaa, onko palvelimella x264.exe tiedostoa. Jos tiedostoa ei löydy, ohjelma sulkeutuu. x264.exe tiedoston löytyessä ohjelman tulostaa alkuvalikon. Alkuvalikosta on mahdollista muuttaa asiakaskoneen asetuksia valitsemalla A.

Asetuksista voidaan määrittää palvelimen osoite ja portti, johon asiakaskone ottaa yhteyden. Asetuksista voi määrittää myös, mitä asiakaskone tekee väliaikaistiedostoille ohjelman sulkeuduttua: poistetaanko ne vai säilytetäänkö ne.

Asetuksien määrittämisen jälkeen takaisin alkuvalikkoon pääsee kirjoittamalla T ja asiakasohjelma käynnistetään V kirjaimella, jolloin asiakaskone yrittää yhdistää palvelimeen. Jos palvelin ei vastaa, ohjelma suljetaan.



Kuva 7. Palvelimen ja asiakkaan välinen keskustelu. Tiedoston lähetystä edeltävä keskustelu ei ole molemmilla kerroilla sama johtuen ohjelmien erilaisesta rakenteesta.

Kun asiakaskone saa muodostettua yhteyden palvelimeen, asiakaskone kysyy heti palvelimelta siirrettävän tiedoston kokoa. Jokaista uutta asiakasta kohden palvelin tekee uuden säikeen. Säikeissä palvelinkone odottaa aina asiakkaan viestiä siitä mitä pitäisi seuraavaksi tehdä. Palvelimen saadessa viestin asiakkaan toiveesta saada siirrettävän tiedoston koko, palvelin lähettää tiedostokoon asiakaskoneelle. Tämän jälkeen asiakaskone kertoo palvelimelle olevansa valmis itse tiedoston vastaanottoon. Palvelimen saadessa viestin asiakkaan valmiudesta, palvelin aloittaa tiedoston siirron asiakkaalle. Kun tiedosto on siirretty, asiakaskone aloittaa videon pakkaamisen. Videon pakkaamiseen jälkeen asiakaskone lähettää palvelimelle viestin halukkuudesta lähettää pakattu tiedosto. Samalla asiakaskone lähettää tiedon tiedoston koosta palvelimelle. Jos palvelin ker-

too olevansa valmis, asiakaskone aloittaa pakatun tiedoston lähettämisen palvelimelle (Kuva 7).

Kun asiakaskone on lähettänyt tiedoston, väliaikaistiedosto (palvelimelta saatu pakattava tiedosto) poistetaan (jos asetuksissa niin on määrätty) ja ohjelma lopetetaan.

4. TESTAUS

Ohjelman toiminta on testattu ja todettu kuvausta vastaavaksi. Nopeustestien tarkastelu päätettiin suorittaa teoreettisella tasolla, koska käytännön testejä ei ole mahdollista suorittaa samalla mittakaavalla kuin teoreettisella tasolla tarkasteltaessa.

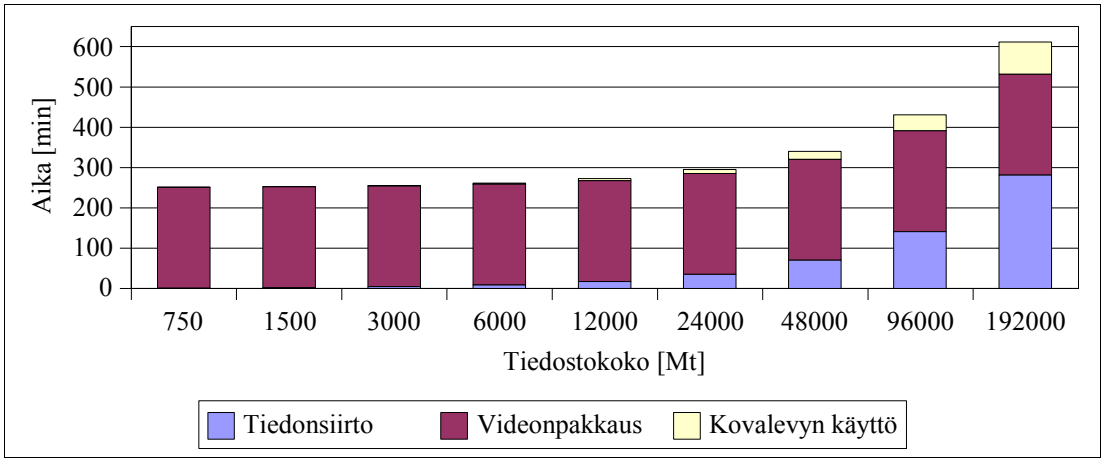
Jokaisessa ohjelmassa tulee jossakin vaiheessa nopeuden kannalta rajoittavat tekijät vastaan. Hallitsevassa asemassa olevaa rajoittavaa tekijää kutsutaan kuvaavasti pullonkaulaksi. Ohjelman nopeuden määräävät käytettävien tietokoneiden nopeudet sekä verkon nopeus.

Seuraavissa esimerkeissä on käytetty seuraavanlaisia arvoja, ellei muuta ole mainittu: videon pituus on 100 minuuttia, videon kehysten määrä sekunnissa on 25, verkon nopeus on 100Mbps ja jokaisen koneen pakkausnopeus on viisi kehystä sekunnissa. Tiedostokoolla tarkoitetaan pakattavan tiedoston kokoa. Pakatun tiedoston kooksi on määrätty kymmenesosa pakattavan tiedoston koosta, mikä on huomioitu myös siirtoon käytettävässä ajassa. Kovalevyn nopeutena on 80Mt/s ja palvelin käsittelee tiedoston kahteen kertaan ennen siirto-operaatioita, sekä lopuksi yhdistää pakatut tiedostot. Yhteensä palvelimen kovalevylle tulee siis 2,1kertainen määrä käsiteltävää tietoa alkuperäiseen tiedostokokoon verrattuna.

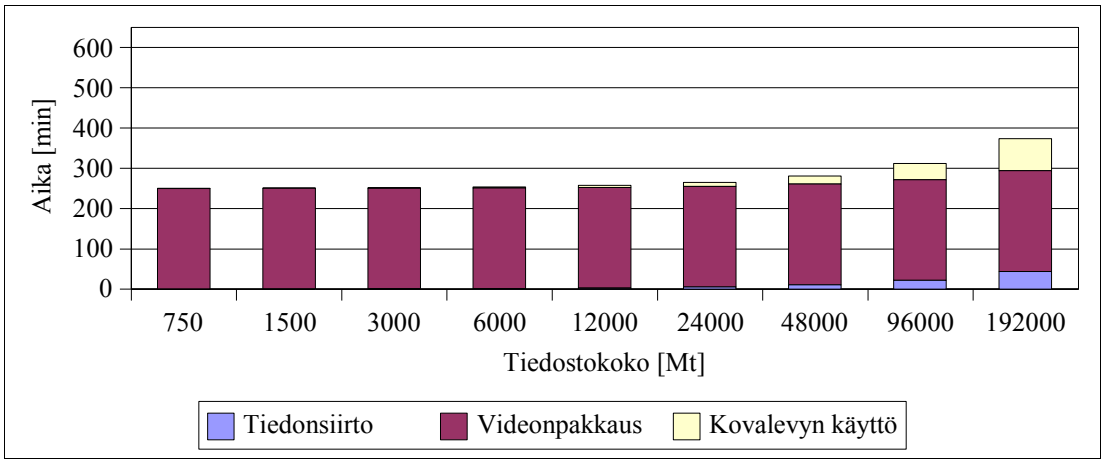
Tiedonsiirtoon kuluva aika koituu jossain vaiheessa pullonkaulaksi siirrettäessä suuria tietomääriä (Kuva 8). Tiedostokoon muuttumisella tarkoitetaan ainoastaan pakattavan tiedoston bittinopeuden muuttumista, jolloin pakattava tietomäärä kehyksinä pysyy samana. Kun käytössä on viisi konetta, pakkaukseen kuluva aika pienenee, mutta tiedonsiirto ja kovalevyn käyttö pysyy samana (Kuva 10). Siirryttäessä 20 koneen käyttöön,

muuhun kuin pakkaukseen kuluvan ajan merkitys kasvaa merkittävästi ja ohjelman nopeuden kannalta pullonkaulaksi muodostuu tiedonsiirto sekä kovalevyn nopeus(Kuva 12).

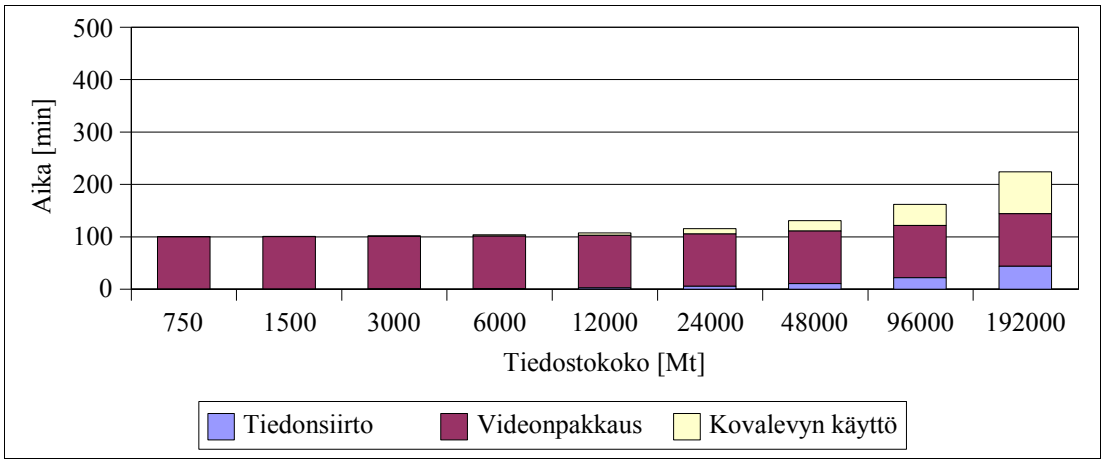
Jos verkon nopeus kasvatetaan 1Gbps:iin, niin kahdella koneella pakattaessa videonpakkaukseen kuluu eniten aikaa ja muita hidastavia tekijöitä ei ehdi syntymään (Kuva 9). Palvelin ei pysty hyödyntämään verkon tuomaa nopeuden lisää täydellisesti johtuen kovalevyn nopeudesta (80Mt/s tai 640Mbps). Näin ollen 1Gbps verkosta palvelin pystyy hyödyntämään vain osan. Viidellä koneella verkon nopeuttaminen auttaa vielä hyvin (Kuva 11), mutta 20 koneella verkon nopeutta nostamalla pullonkauloja ei saada enää poistettua (Kuva 13).



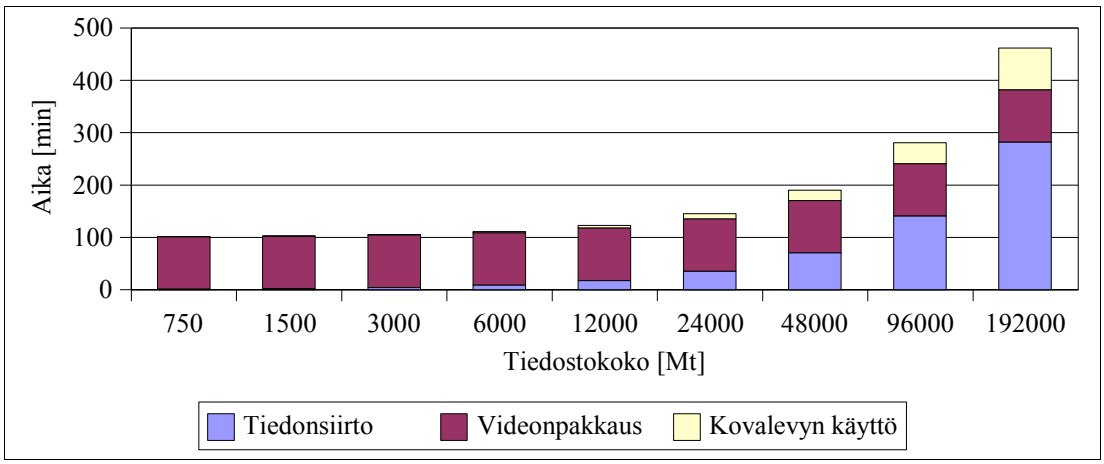
Kuva 8. Toimintoihin kulunut aika kahdella koneella ja 100Mbps verkkoyhteydellä.



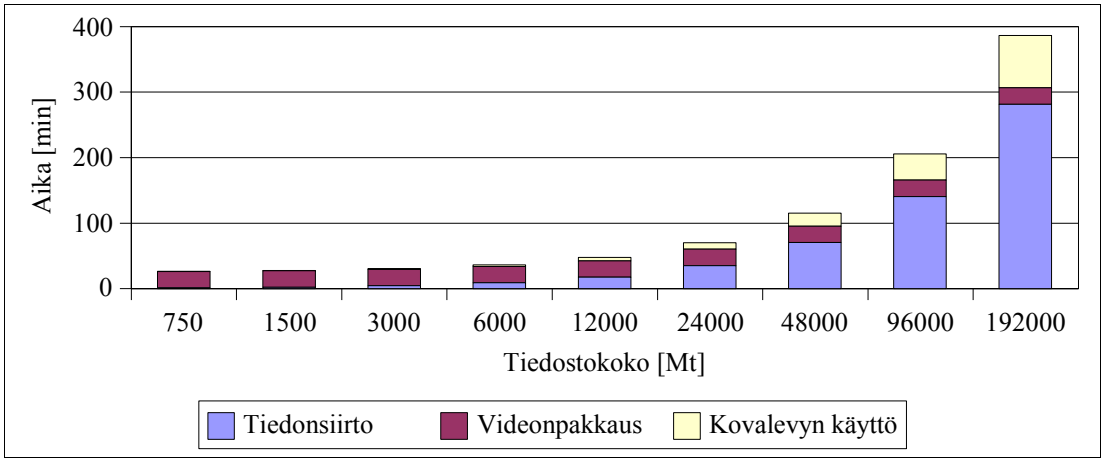
Kuva 9. Toimintoihin kulunut aika kahdella koneella ja 1Gbps verkkoyhteydellä.



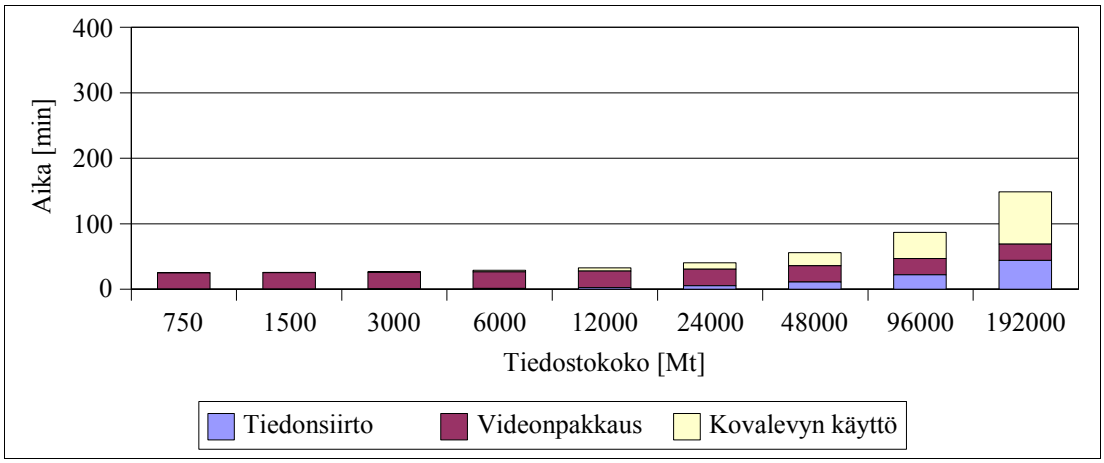
Kuva 10. Toimintoihin kulunut aika viidellä koneella ja 100Mbps verkkoyhteydellä.



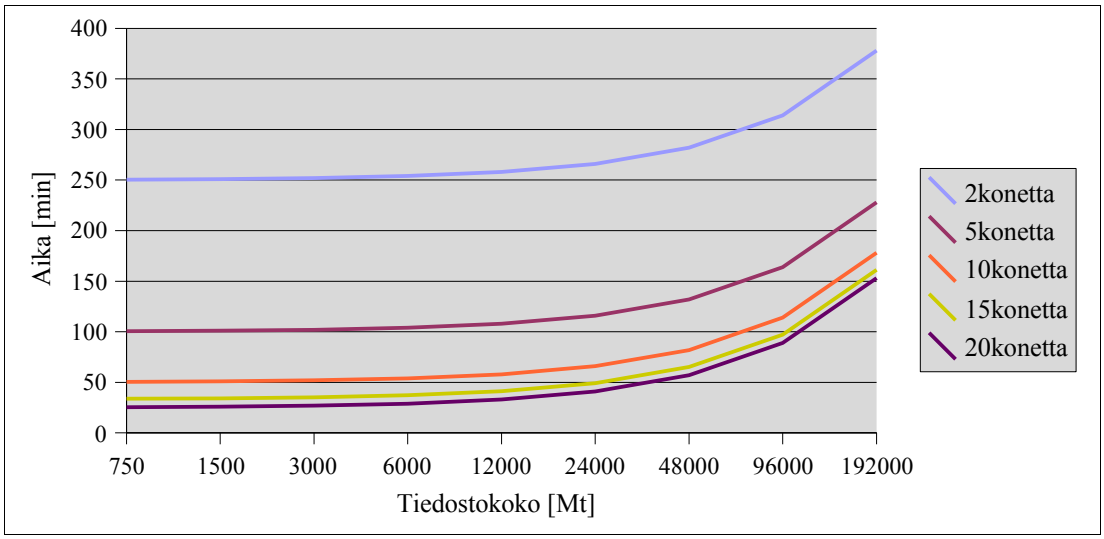
Kuva 11. Toimintoihin kulunut aika viidellä koneella ja 1Gbps verkkoyhteydellä.



Kuva 12. Toimintoihin kulunut aika 20 koneella ja 100Mbps verkkoyhteydellä.



Kuva 13. Toimintoihin kulunut aika 20 koneella ja 1Gbps verkkoyhteydellä.



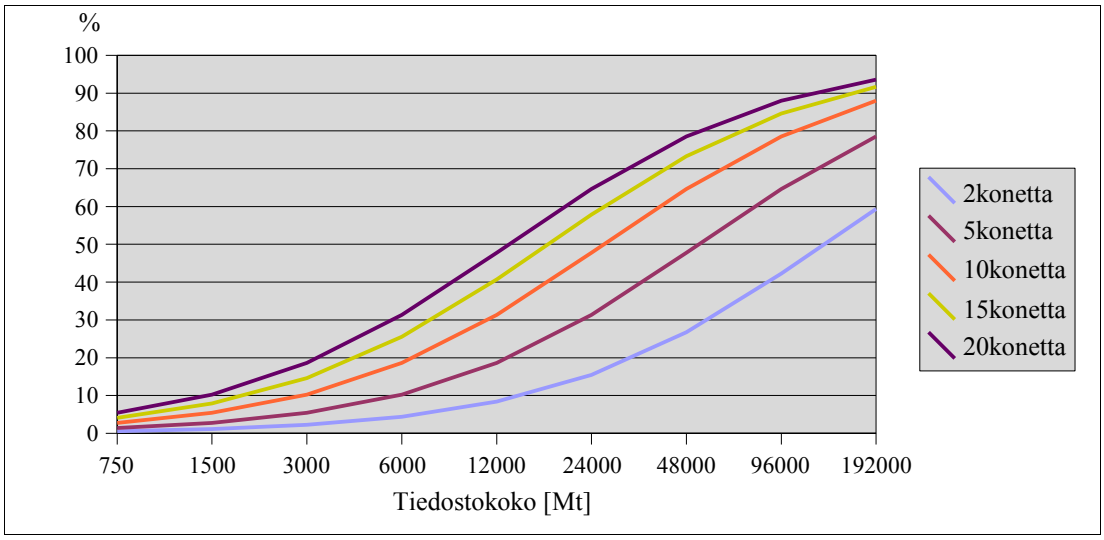
Kuva 14. Toimintoihin kulunut aika yhteensä, kun käytössä on 1Gbps verkko.

Vaikka koneiden lukumäärän lisääminen nopeuttaakin hieman pakkaamista suuria tiedostoja käsiteltäessä, sillä ei ole kuitenkaan mitään merkitystä, koska muut asiat hidastavat paljon enemmän prosessia (Kuva 14).

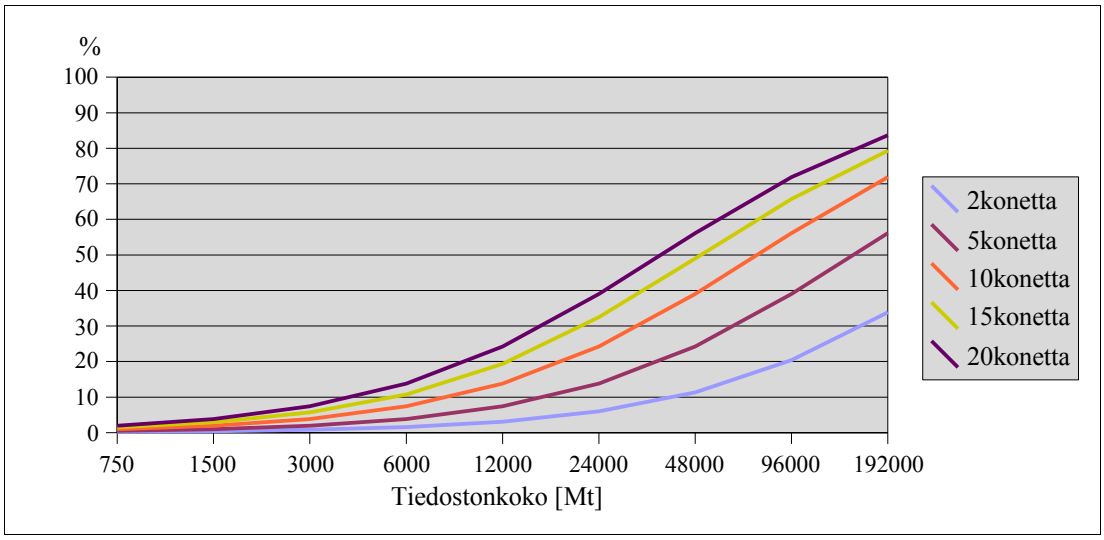
On käyttäjästä kiinni, kuinka paljon aikaa hän haluaa kuluttaa muuhun kuin videonpakkaamisen, mutta selvät rajat on tehtävä hyödyn maksimoimiseksi (Kuva 15). 30%:a on hyvä pitää ns. ”kipurajana”, jolloin esim. kahdella koneella pakattaessa 100Mbps verkon kautta voi siirtää n. 60Gt:a pakkausajan pysyessä vielä 70%:ssa kokonaisprosessin ajasta. 20:ta konetta käytettäessä pakkaukseen kuluva aika on pienentynyt verrattaessa kahdella koneella suoritettavaan pakkaukseen, joten palvelimen on suotavaa siirtää vain n. 6Gt:a, jottei pullonkaulaksi muodostuisi riittämätön verkkokapasiteetti. Verkon no-

peutta kasvattamalla 1Gbps:iin 20 koneella voi jo käsitellä yli kaksi kertaa suurempia videotiedostoja (Kuva 16).

Verkon nopeutta ei siis pystytä enää kasvattamaan yli 1Gbps, koska palvelin ei pysty käsittelemään tiedostoja niin nopeasti eikä palvelimen kovalevyn nopeutta voida nostaa ilman monen kovalevyn RAID järjestelmiä. Koneiden prosessoritehoa voidaan kasvattaa, mutta sillä saamme vain sen aikaan, että pakkaukseen kuluvan ajan osuus kokonaisajasta pienenee (Kuva 17).

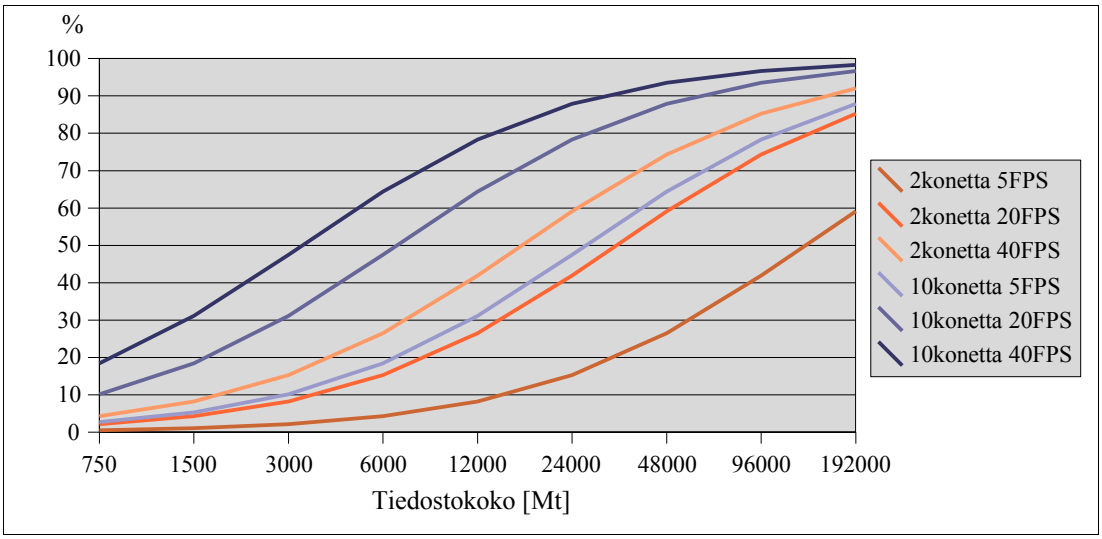


Kuva 15. Kovalevyn toimintoihin sekä tiedonsiirtoon kuluvan ajan osuus prosentteina 100Mbps verkossa.



Kuva 16. Kovalevyn toimintoihin sekä tiedonsiirtoon kuluvan ajan osuus prosentteina 1Gbps verkossa.

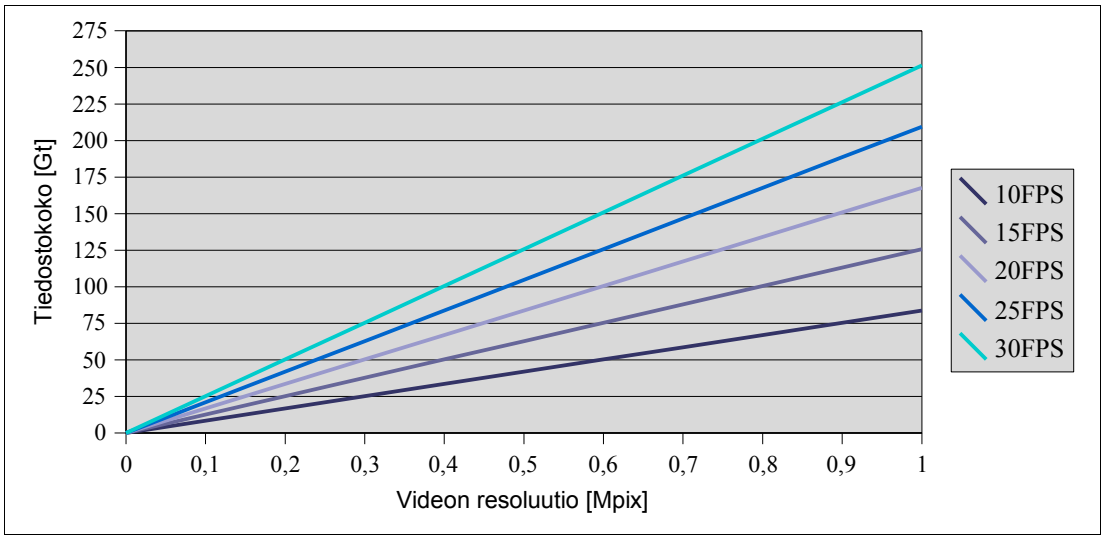
Ohjelman päätarkoitus on videonpakkaamiseen käytettävän ajan lyhentäminen, joten kokonaisajasta videonpakkaukseen kulunut aika olisi hyvä pitää korkeana, jotta ohjelmasta saataisiin suurin hyöty. Tämä tapahtuu käytettäessä pieniä tiedostokokoja ja nopeaa verkkoa. Videonpakkaus pitäisi olla myös mahdollisimman raskasta, eli pakkaukseen osallistuvien koneiden kehysmäärä sekunnissa (FPS) pitäisi olla alhainen. Videonpakkaus onkin jo raskasta, koska käytössä on MPEG-4 AVC koodekki, joten olisi vielä suositeltavaa käyttää koodekista raskaimpia (parhaimpia) laatuasetuksia.



Kuva 17. Kovalevyn toimintoihin sekä tiedonsiirtoon kuluvan ajan osuus prosentteina 100Mbps verkossa.

5. YHTEENVETO

Hiemankin suuremmat pakkaamattomassa muodossa olevat .y4m tiedostot saattavat vaatia jopa monta sataa gigatavua tallennustilaa (Kuva 18). Pakkaamattoman videotiedoston hajauttaminen monelle koneelle koituu nopeasti prosessin pullonkaulaksi. Vaikka häviöllisesti pakatun tiedoston uudelleenpakkaaminen ei olekaan suositeltavaa, uudelleenpakkaus ei asettaisi kovia vaatimuksia verkolle ja kovalevylle johtuen paljon pienemmästä tiedostokoosta. Joten ohjelmaan voisi kehitellä tuen myös erilaisille häviöllisesti pakatuille videotiedostoille.



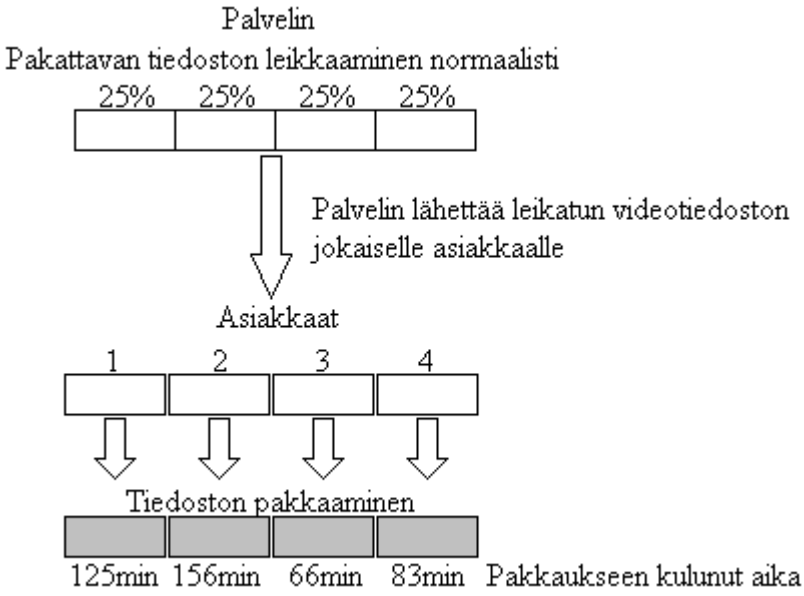
Kuva 18. 100minuutin .y4m tiedostokoko gigatavuina erilaisilla FPS lukemilla (huom. esim. DVD:n resoluutio 720*576 megapikseleinä on vain 0,4Mpix)

Graafinen käyttöliittymä olisi myös hyvä toteuttaa. Se helpottaisi osaltaan ohjelman käyttöä ja mahdollistaisi paremmat tilastot ohjelman toiminnasta. Graafiseen käyttöliittymään voisi tehdä myös reaaliaikaisen keskustelualueen, jossa käyttäjät voisivat lähettää toisilleen viestejä. Keskustelualueella palvelinkoneella olisi mahdollisuus poimia käyttäjät, jotka haluavat osallistua videonpakkaamiseen. Käyttäjillä voisi olla mahdollisuus valita oma valmiustilansa, joka määrittäisi käyttäjän halukkuuden osallistua pakkaamiseen.

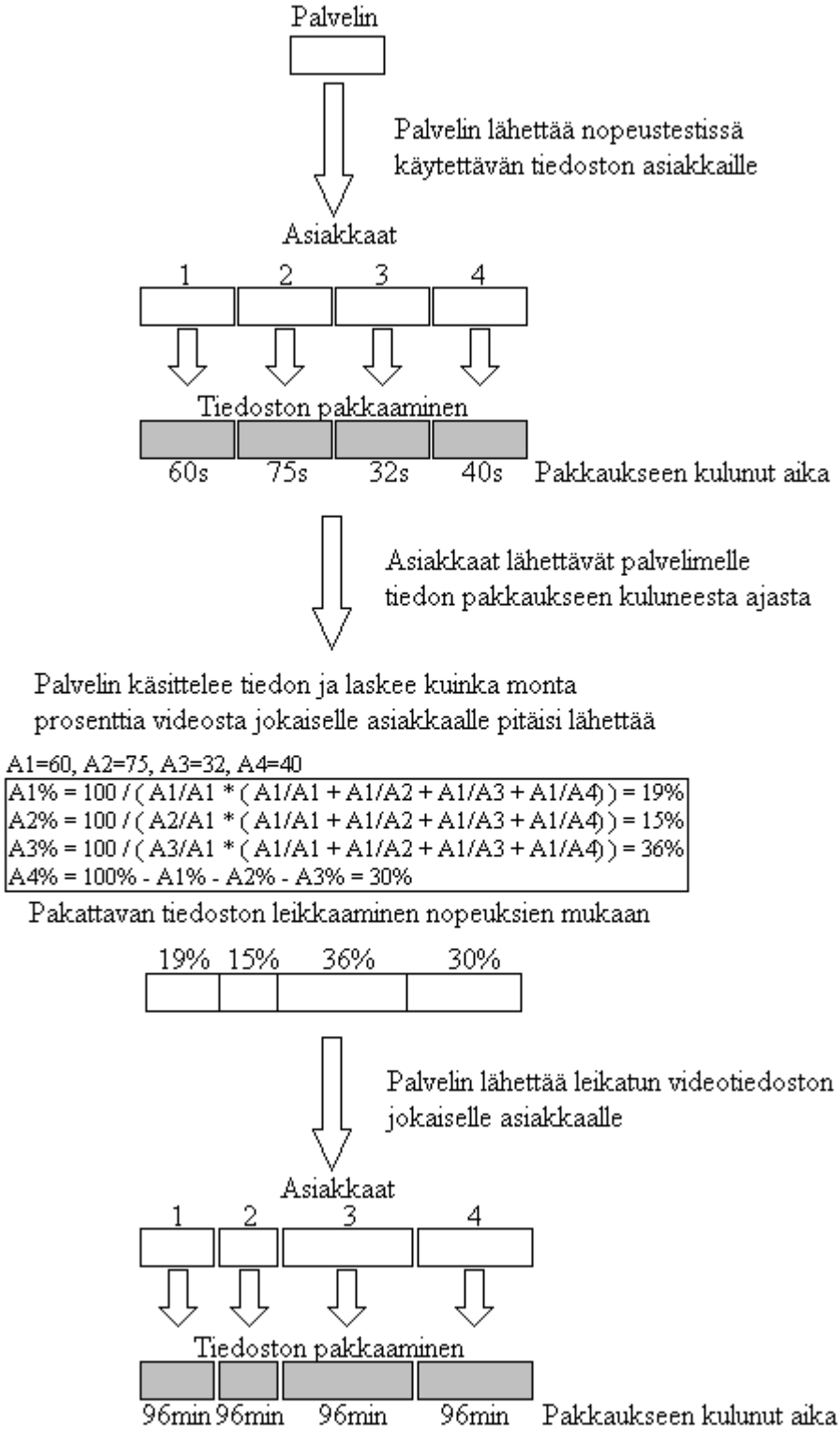
Ohjelmaan olisi hyvä tehdä mahdollisuus jatkaa pakkausprosessia, jos esim. yhteys johonkin koneeseen katkeaa kesken prosessin. Muunlaisten virhetilanteiden varalle olisi myös suositeltavaa kehittää erilaisia toipumismenetelmiä.

Ohjelmaan tulisi kehittää parempi pakettien lähetysmenetelmä. Nykytilassaan palvelin lähettää kaikille yhtä suuret leikatut videotiedostot, joten ohjelman nopeuden määrää kaikkein hitain kone (Kuva 19). Jos asiakaskoneita olisi viisi ja neljä näistä suoriutuisi pakkauksesta neljässä minuutissa, mutta yhdellä asiakkaalla kestäisi kymmenen minuuttia, ohjelman nopeuden pullonkaulaksi muodostuisi hitain kone. Videotiedostot pitäisi jakaa niin, että nopein kone saisi suurimman palan tiedostosta ja hitain kone saisi pienimmän palan, jolloin hitaimman ja nopeimman koneen pakkausnopeus olisi sama. Tällainen ratkaisu vaatisi koneiden nopeuden määrittämistä palvelimen asetuksista. Eri-

laisten koneiden keskinäisiä nopeuseroja saattaa olla lähes mahdoton määrittää. Helpoin ja varmin tapa olisi ennen varsinaista pakkausta suorittaa nopeudentestaus jokaisella asiakaskoneella. Palvelin lähettäisi kaikille asiakkaille pienikokoisen videoleikkeen, jonka asiakaskoneet pakkaisivat. Pakkausajan perusteella palvelin määrittäisi kuinka suuren osan varsinaisesta videosta se jakaisi millekin asiakkaalle (Kuva 20). Nopeudentestaukseen tulisi myös ottaa huomioon asiakkaan yhteyden nopeus nopeuden ollessa sellainen, joka saattaisi merkittävästi vaikuttaa prosessiin.



Kuva 19. Pakattavan tiedoston leikkaaminen normaalisti. Pakattavan videon kehyksien määrä on 150000. Esimerkissä käytetään samoja asiakaskoneita kuin kuva 20:ssä.



Kuva 20. Pakattavan tiedoston jakaminen asiakaskoneiden nopeuksien perusteella. Pakattavan videon kehyksien määrä on 150000 ja nopeustestissä käytettävän videon kehyksien määrä on 300.

LÄHTEET

1. Wikipedia-projektin osanottajat. 25.10.2006. Chroma subsampling. [online] Wikipedia. [Viitattu 26.10.2006]. Saatavissa: http://en.wikipedia.org/w/index.php?title=Chroma_subsampling&oldid=83575650

2. Iain E. G. Richardson. H.264 and MPEG-4 video compression. 1st ed. Great Britain: Wiley, 2003. 281 s.

3. Divx Advanced Research Team (DARC). 2004. The Official Divx Guide 5.2.1.1. [online]. DigitAl56k/DivxNetworks. [Viitattu 29.10.2006]. Saatavissa: http://download.divx.com/divx/DivXUserGuide521-en.exe

4. Keränen, Lamberg, Penttinen. Digitaalinen Media. 1st ed. Porvoo: Docendo, 2005. 398 s.

5. AfterDawn.com sisällöntuottajat. 2006. Multi-pass pakkaus. [online]. AfterDawn Oy. [viitattu 29.10.2006]. Saatavissa: http://fin.afterdawn.com/sanasto/termit/multipass.cfm

6. Wikipedia-projektin osanottajat. 15.10.2006. Python. [online]. Wikipedia. [Viitattu 20.10.2006]. Saatavissa: http://fi.wikipedia.org/w/index.php?title=Python&oldid=1749782

7. Doom9.net sisällöntuottajat. 2005. Codec shoot-out 2005 – Final. [online] Doom9.net. [Viitattu 21.10.2006] Saatavissa: http://www.doom9.org/index.html?/codecs-final-105-1.htm

8. MultimediaWiki-projektin osanottajat. 17.9.2006. YUV4MPEG2. [online]. MultimediaWiki. [Viitattu 24.10.2006] Saatavissa: http://wiki.multimedia.cx/index.php?title=YUV4MPEG2&oldid=5997

9. Kannisto, O. Videot näkyviin koodekeilla. Mikrobitti. Vol. 9/2006, s. 58-61.

10. GridIron Software. 17.4.2005. GridIron and Microsoft Unveil GridIron X-Factor for Microsoft Windows Media Encoder. [online]. GridIron Software. [viitattu 6.11.2006] Saatavissa: http://www.gridironsoftware.com/news/Default.asp?Page=News_xfactor_wme

11. Wikipedia-projektin osanottajat. 18.3.2006. Kuvanpakkaus. [online]. Wikipedia. [Viitattu 10.12.2006]. Saatavissa: http://fi.wikipedia.org/w/index.php?title=Kuvanpakkaus&oldid=909150

PALVELIN.PY OHJELMAKOODI

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*- #Mahdollistaa mm. ääkköset.

import os, sys        #Tiedoston käsittely, x264 ohjelman käynnistys, ym
import time           #Ajan mittaamiseen, toimintojen nopeus saadaan laskettua
import threading      #Mahdollistaa säikeistäminen
from socket import *  #Socket toiminnot

########### VIDEON PAKKAAMINEN ########################################

class encoding ( threading.Thread ):

   def run ( self ):

      print "Aloitetaan videonpakkaus."
      t1 = time.clock()
      command = "x264 --subme 7 --progress -o video000.264 tempfile000.y4m"
      print command
      os.system(command)
      t2 = time.clock()
      nopeus = t2 - t1  #Lasketaan pakkaukseen kulunut aika sekunteina
      print ""
      print "Videonpakkaus valmis, aikaa kului " + str(round(nopeus, 2)) + "
      sekuntia."

########## </VIDEON PAKKAAMINEN LOPPUI> ##############################

########## PALVELIN-ASIAKAS KOMMUNIKOINTI ############################

class new_connection ( threading.Thread ):

   def run (self):

      global connection
      global address
      global file_identify
      global client_valmis

      file_identify_x = file_identify
      connection_x = connection
      address_x = address

      tiedoston_koko = 0
      filesize = 0

      print 'PALVELIN-ASIAKAS%d: yhteys luotu.' % file_identify_x

      tiedoston_koko = os.stat('tempfile%03d.y4m' % file_identify_x) #Luetaan
      siirrettävän tiedoston koko
      filesize = tiedoston_koko.st_size

      lahetettava_filu = os.open('tempfile%03d.y4m' %
      file_identify_x,os.O_BINARY)

      while 1:

         data = connection_x.recv(1024) #Vastaanotetaan asiakkaan viesti

         if data == "size":#Lähetetään asiakkaalle lähetettävän tiedoston koko
            connection_x.send(str(filesize))
```

```
            elif data == "ready":    #Lähetetään asiakkaalle itse tiedosto

               print ""
               print "PALVELIN-ASIAKAS%d: L\x84hetet\x84\x84n asiakkaalle
               tempfile%03d.y4m tiedosto." % (file_identify_x,file_identify_x)
               print "PALVELIN-ASIAKAS%d: Tiedoston koko:" % file_identify_x ,
               filesize / 1024, "kilotavua."
               print ""

               while 1:
                  buf = os.read(lahetettava_filu, 8192)

                  if not buf:   #Suljetaan lähetys kun bufferi on tyhjentynyt
                     print "PALVELIN-ASIAKAS%d: Tiedosto l\x84hetetty." %
                     file_identify_x
                     os.close(lahetettava_filu)
                     break

                  connection_x.send(buf)

            elif data == "takaisinlahetys":

               filesize = 0
               filesize = connection_x.recv(1024)

               vastaus = "ready"
               connection_x.send(vastaus)

               print ""
               print "PALVELIN-ASIAKAS%d: Vastaanotetaan pakattua video%03d.264
               tiedostoa." % (file_identify_x,file_identify_x)
               print ""

               out_file = open("video%03d.264" % file_identify_x,"wb")

               buf = 0
               data_count = 0

               while 1:

                  buf = connection_x.recv(8192)           #Siirtobufferin koko
                  data_count += len(buf)

                  out_file.write(buf)  #Kirjoitetaan bufferin sisälto kovalevylle

                  if data_count == int(filesize):

                     print "PALVELIN-ASIAKAS%d: Vastaanotettu" % file_identify_x,
                     data_count / 1024, "kilotavua."

                     print 'PALVELIN-ASIAKAS%d: Tiedosto video%03d.264
                     vastaanotettu, koko' % (file_identify_x,file_identify_x),
                     data_count, "tavua."

                     out_file.close()
                     connection_x.close()     #Sulje yhteys
                     client_valmis += 1       #Lasketaan kuinka monta asiakasta
                                              #on suorittanut videonpakkauksen
                     break

               break

##########  </PALVELIN-ASIAKAS KOMMUNIKOINTI LOPPUI> ##################
```

```
########## OHJELMAN ALKUVALIKKO ######################################

class main_app ( threading.Thread ):

   def run ( self ):

      global connection
      global address
      global file_identify
      global client_valmis

      server_encoding = 1
      tempfile_poisto = False
      client_lkm = 2
      video_file = "example.y4m"
      output_file = "output.264"
      server_port = "2000"
      valinta = ""
      valinta_x = ""

      x264_exist = os.path.isfile("x264.exe") #Tarkistetaan löytyykö x264.exe

      if x264_exist == False:
         print "x264.exe tiedostoa ei l\x94ydetty, suljetaan ohjelma."
         sys.exit()

      while valinta != "v" and valinta != "V":

         print ""
         print "+-----------------------------------------------------+"
         print "|   Hajautettu videonpakkausohjelmisto - Palvelin     |"
         print "+-----------------------------------------------------+"
         print "| Valitse A jos haluat muuttaa asetuksia              |"
         print "| Valitse V jos olet valmis aloittamaan pakkaamisen |"
         print "+-----------------------------------------------------+"

         valinta = raw_input("\n")

         if valinta == "a" or valinta == "A":

            while valinta != "t" and valinta != "T":

               print ""
               print "Nykyiset asetukset:"

               if server_encoding == 1:
                  print "[1] Palvelin osallistuu pakkaamiseen: Kyll\x84."
               if server_encoding == 0:
                  print "[1] Palvelin osallistuu pakkaamiseen: Ei."

               print "[2] Asiakaskoneiden lukum\x84\x84r\x84:", client_lkm
               print "[3] Pakattava tiedosto:", video_file
               print "[4] Tallennetaan nimell\x84:", output_file

               if tempfile_poisto == True:
                  print "[5] V\x84liaikaistiedostojen poisto: Kyll\x84."
               if tempfile_poisto == False:
                  print "[5] V\x84liaikaistiedostojen poisto: Ei."

               print "[6] Palvelimen portti:", server_port

               print "\nValitse numero, jonka asetusta haluat muuttaa,"
               print "tai valitse T palataksesi takaisin."

               valinta = raw_input("\n")

               if valinta == "1":
```

```
   while 1:

      valinta_x = raw_input("Haluatko palvelimen osallistuvan
      pakkaamiseen? [K]yll\x84 tai [E]i:")

      if valinta_x == "k" or valinta_x == "K":
         server_encoding = 1
         break

      elif valinta_x == "e" or valinta_x == "E":

         if int(client_lkm) == 1:
            print ""
            print "Jos et aio k\x84ytt\x84\x84 palvelinta
            tiedoston pakkaamiseen"

            print "ja sinulla on k\x84yt\x94ss\84si vain yksi
            asiakaskone, et tarvitse t\84t\84 ohjelmaa!"
            print ""

            print "Vaihda asiakaskoneiden m\84\84r\84ksi 2 tai
            enemm\84n, niin sitten palvelimen ei tarvitse
            osallistua pakkaukseen."
            print ""

         if int(client_lkm) != 1:
            server_encoding = 0
            break

      print "\nV\x84\x84r\x84 vastaus! Valitse uudestaan!"
      print "Kirjoita K (Kyll\x84) tai E (Ei)\n"

if valinta == "2":

   while 1:

      client_lkm = raw_input("Anna asiakaskoneiden
      lukum\x84\x84r\x84:")

      if int(server_encoding) == 0 and int(client_lkm) == 1:
         print ""
         print "Jos et aio k\x84ytt\x84\x84 palvelinta
         tiedoston pakkaamiseen"

         print "ja sinulla on k\x84yt\x94ss\84si vain yksi
         asiakaskone, et tarvitse t\84t\84 ohjelmaa!"
         print ""

         print "Aseta palvelin osallistumaan pakkaukseen, niin
         sitten voit m\x84\x84rittää asiakaskoneiden
         lukum\x84\x84r\x84ksi 1"
      else:
         break

if valinta == "3":

   while 1:
      video_file = raw_input("Anna pakattavan tiedoston nimi:")

      file_exist = os.path.isfile(video_file) #Tarkistaa onko
      tiedosto olemassa

      if file_exist == True:
         break

      else:
         print "Tiedostoa", video_file, "ei ollut olemassa,
```

```
                        tarkista tiedoston nimi."

                if valinta == "4":

                   while 1:
                      output_file = raw_input("Valmis pakattu tiedosto
                      tallennetaan nimell\x84:")
                      break

                if valinta == "5":

                   while 1:

                      valinta_x = raw_input("Haluatko ett\x84
                      v\x84liaikaistiedostot poistetaan lopuksi? [K]yll\x84 tai
                      [E]i:")

                      if valinta_x == "k" or valinta_x == "K":
                         tempfile_poisto = True
                         break

                      elif valinta_x == "e" or valinta_x == "E":
                         tempfile_poisto = False
                         break

                      print "\nV\x84\x84r\x84 vastaus! Valitse uudestaan!"
                      print "Kirjoita K (Kyll\x84) tai E (Ei)\n"

                if valinta == "6":

                   while 1:
                      server_port = raw_input("Kirjoita k\x84ytett\x84v\x84
                      portti:")
                      break

########## </OHJELMAN ALKUVALIKKO LOPPUI> ############################

########## VIDEON ALKUTUNNISTEEN TARKISTUS ############################

      tiedoston_tarkistus = os.open(video_file,os.O_BINARY)

      buf = 0
      video_y4m = False

      while 1:
         buf = os.read(tiedoston_tarkistus, 8192)

         header_start = buf.find('YUV4MPEG2')

         if header_start == 0:
            video_y4m = True
            os.close(tiedoston_tarkistus)
            break

         if header_start != 0:
            video_y4m = False
            os.close(tiedoston_tarkistus)
            break


########## </VIDEON ALKUTUNNISTEEN TARKISTUS LOPPUI> #################

########## KEHYSTEN ETSINTÄ ###########################################

      if video_y4m == True:

         print "Tiedosto on y4m videotiedosto."
         print "Etsit\x84\x84n tiedoston framet."
```

```
            temppi_filu = os.open(video_file,os.O_BINARY)

            framelista = [0]*131072

            header_tiedot = 0
            frame_start = 0
            position = 0
            count = 0
            buf = 0

            while 1:

               buf = os.read(temppi_filu, 8192)

               if not buf:
                  os.close(temppi_filu)
                  break

               frame_start = buf.find('FRAME')

               if header_tiedot == 0:
                  header_tiedot = buf[0:frame_start]

               while frame_start != -1:
                  count += 1
                  start = frame_start + 1
                  framelista[count-1] = frame_start + position
                  frame_start = buf.find('FRAME', start)

               position += len(buf)

            print "Yhteens\x84 %d kehyst\x84." % count

##########  </KEHYSTEN ETSINTÄ LOPPUI> ################################

##########  LEIKKAUSKOHTIEN TAULUKOINTI ################################

            print "Videon alkutunniste: ", header_tiedot

            tiedostojen_lkm = int(server_encoding) + int(client_lkm)

            print "Paloitellaan videotiedosto", tiedostojen_lkm, "eri osaan."

            puolituskohdat = [0]*32

            osan_pituus = float(count)/tiedostojen_lkm    #Laskee kuinka monta
                                        #framea yhdessa paloitellussa
                                        #tiedostossa tulee olemaan.

            index = 0
            leikkauskohta = 0
            while index != tiedostojen_lkm:

                leikkauskohta = osan_pituus * index #Leikkauskohta = Framen
                                                    #numero joka on leikkauskohdassa
                puolituskohdat[index] = framelista[int(leikkauskohta)]   #Hakee
                                        #framelistasta leikkauskohdassa olevan
                                        #framen sijainnin ja tallentaa sen
                                        #taulukkoon

                index += 1

##########  </LEIKKAUSKOHTIEN TAULUKOINTI LOPPUI> ####################

##########  VIDEON PALOITTELU ########################################

            index = 0
```

```
            while index != tiedostojen_lkm:

               out_file = 0
               cutting = 0
               position = 0
               buf = 0

               temppi_filu = os.open(video_file,os.O_BINARY)
               out_file = open("tempfile%03d.y4m" % index ,"wb")

               os.lseek(temppi_filu, puolituskohdat[index], 0)

               out_file.write(header_tiedot)

               t_leikkuu1 = time.clock()

               if index == tiedostojen_lkm -1:        #Viimeinen tiedosto...
                                #...tallennetaan eritavalla (tiedostojen_lkm -1)

                  buf = 0
                  while 1:

                     buf = os.read(temppi_filu, 131072)

                     if not buf:
                        out_file.close()
                        os.close(temppi_filu)
                        break

                     out_file.write(buf)

               else:  #Käytetaan osien leikkaamiseen (paitsi ei viimeiselle)

                  position = puolituskohdat[index]
                  while 1:

                     buf = os.read(temppi_filu, 131072) #128kt:n buffer käytossä

                     if not buf:
                        out_file.close()
                        os.close(temppi_filu)
                        break

                     position += len(buf)

                     if position > puolituskohdat[index+1]:
                        cutting = 131072 - (position - puolituskohdat[index+1])
                        out_file.write(buf[:cutting])
                        out_file.close()
                        os.close(temppi_filu)
                        break

                     out_file.write(buf)

               t_leikkuu2 = time.clock()
               nopeus = t_leikkuu2 - t_leikkuu1
               print "tempfile%03d.y4m tallennettu" % index, str(round(nopeus,
                      2)), "sekunnissa."

               index += 1

            print "Videotiedosto paloiteltu."
            print ""
##########  </VIDEON PALOITTELU LOPPUI> ###############################

            if server_encoding == 1:                     #Aloitetaan pakkaaminen
               encoding().start()
```

```
########## SOCKET KOODI - PALVELIN ###################################
         myHost = ''
         myPort = int(server_port)                      #Palvelimen portti

         s = socket(AF_INET, SOCK_STREAM)               #Socketin luominen
         s.bind((myHost, myPort))                       #Porttiin bindaaminen
         s.listen(20)                    #Sallii 20 samanaikaista yhteyttä

         if server_encoding == 1:
            file_identify = 0
            client_valmis = 0

         if server_encoding == 0:
            file_identify = -1
            client_valmis = -1

         while file_identify != tiedostojen_lkm -1:

             print "PALVELIN: Odotetaan lisaa asiakkaita..."

             connection, address = s.accept()
             file_identify += 1
             new_connection().start()


         print "Kaikki asiakkaat ovat ottaneet yhteytt\x84."

         while 1:

            if client_valmis != tiedostojen_lkm -1:
               time.sleep(1)
            else:
               break

         print "Tiedostot vastaanotettu.\n"

########## </SOCKET KOODI - PALVELIN LOPPUI> ##########################

########## PAKATTUJEN TIEDOSTOJEN YHDISTÄMINEN ########################

         print "Yhdistet\x84\x84n pakatut videotiedostot."

         index = 0
         buf = 0

         out_file = open(output_file,"wb")

         while index != tiedostojen_lkm:

            temppi_filu = os.open("video%03d.264" % index ,os.O_BINARY)

            while 1:

               buf = os.read(temppi_filu, 131072)

               if not buf:
                  os.close(temppi_filu)
                  break

               out_file.write(buf)

            index += 1

         out_file.close()

         print "Tiedostot yhdistetty."

########### </PAKATTUJEN TIEDOSTOJEN YHDISTÄMINEN LOPPUI> ############
```

```
########### VÄLIAIKAISTIEDOSTOJEN POISTAMINEN ########################

         if tempfile_poisto == True:  #Poistetaan väliaikaistiedostot

            index = 0
            while index != tiedostojen_lkm:

               os.remove("video%03d.264" % index)
               os.remove("tempfile%03d.y4m" % index)
               index += 1

            print "V\x84liaikaistiedostot poistettu."

########### </VÄLIAIKAISTIEDOSTOJEN POISTAMINEN LOPPUI> ##############

      else:
         print "Tiedosto ei ole YUV4MPEG2 videotiedosto"

main_app().start() #Ohjelman käynnistys
```

ASIAKAS.PY OHJELMAKOODI

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*-           #Mahdollistaa mm. ääkköset

import os, sys        #Tiedoston käsittely, x264 ohjelman käynnistys, ym
import time           #Ajan mittaamiseen, toimintojen nopeus saadaan laskettua
from socket import *  #Socket toiminnot

#### ALKUVALIKKO ######################################################

server_add = "localhost"
server_port = "2000"
tempfile_poisto = False
valinta = ""

x264_exist = os.path.isfile("x264.exe")  #Tarkistetaan löytyykö x264.exe
                                         #tiedosto

if x264_exist == False:
   print "x264.exe tiedostoa ei l\x94ydetty, suljetaan ohjelma."
   sys.exit()

while valinta != "v" and valinta != "V":

   print ""
   print "+-------------------------------------------------------------+"
   print "|        Hajautettu videonpakkausohjelmisto - Asiakas         |"
   print "+-------------------------------------------------------------+"
   print "| Valitse A jos haluat muuttaa asetuksia                      |"
   print "| Valitse V jos olet valmis ottamaan palvelimeen yhteytt\x84    |"
   print "+-------------------------------------------------------------+"

   valinta = raw_input("\n")

   if valinta == "a" or valinta == "A":

      while valinta != "t" and valinta != "T":
         print "\nNykyiset asetukset:"
         print "[1] Palvelimen osoite:", server_add
         print "[2] Palvelimen portti:", server_port
         if tempfile_poisto == True:
            print "[3] V\x84liaikaistiedostojen poisto: Kyll\x84."
         if tempfile_poisto == False:
            print "[3] V\x84liaikaistiedostojen poisto: Ei."

         print "\nValitse numero, jonka asetusta haluat muuttaa,"
         print "tai valitse T palataksesi takaisin."

         valinta = raw_input("\n")

         if valinta == "1":
            server_add = raw_input("Kirjoita palvelimen IP-osoite:")

         if valinta == "2":
            server_port = raw_input("Kirjoita palvelimen portti:")

         if valinta == "3":

            while 1:
               valinta_x = raw_input("Haluatko ett\x84 v\x84liaikaistiedostot
                           poistetaan lopuksi? [K]yll\x84 tai [E]i:")
```

```
                if valinta_x == "k" or valinta_x == "K":
                   tempfile_poisto = True
                   break

                elif valinta_x == "e" or valinta_x == "E":
                   tempfile_poisto = False
                   break

                print "\nV\x84\x84r\x84 vastaus! Valitse uudestaan!"
                print "Kirjoita K (Kyll\x84) tai E (Ei)\n"


#### </ALKUVALIKKO LOPPUI> ##############################################

#### SOCKET KOODI - ASIAKAS #############################################

s = socket(AF_INET, SOCK_STREAM)             #TCP socketin luominen

try:
   s.connect((server_add, int(server_port))) #Asiakas yhdistää palvelimeen
except:
   print "Ei saatu yhteytt\x84 palvelimeen."
   sys.exit()

s.send('size')    #Asiakas kysyy palvelimelta siirrettävän tiedoston kokoa

filesize = s.recv(512)  #Vastaanotetaan tieto siirrettävän tiedoston koosta

file_obj = open('client_file.y4m', 'wb')

s.send('ready')         #Asiakas kertoo olevansa valmis vastaanottamaan
t1 = time.clock()       #Lasketaan siirtoon käytettävä aika, aloituskohta
laskin = 0              #Käytetään statistiikan ajastukseen
data_count = 0          #Vastaanotetun datan määrä

while 1:
      laskin += 1
      buf = s.recv(8192)            #Siirtobufferin koko

      data_count += len(buf)

      if laskin == 4:               #Määrätään statistiikan päivitystahti
         print "Vastaanotettu",round(float(data_count) /
                float(filesize)*100,2),"\b%", data_count /
                1024,"kilotavua.","\r",   #\b backspace, \r carriage return
         laskin = 0

      file_obj.write(buf)           #Kirjoitetaan bufferin sisalto kovalevylle

      if data_count == int(filesize):

         print "Vastaanotettu", data_count / 1024, "kilotavua
                -",round(float(data_count) / float(filesize)*100,2),"\b%"
                #\b backspace

         print 'Tiedosto vastaanotettu.\nTiedoston koko', data_count, "tavua."
         file_obj.close()           #Muista sulkea
         break

t2 = time.clock()           #Lasketaan siirtoon kaytettava aika, lopetuskohta
nopeus = t2 - t1            #Lasketaan tiedoston siirtonopeus.
print ""
print "Aikaa kului " + str(round(nopeus, 2)) + " sekuntia."


#### </SOCKET KOODI LOPPUI> ##############################################
```

```
#### VIDEON PAKKAAMINEN ##############################################

print ""
print "Videotiedosto imutettu, aloitetaan pakkaaminen."
print ""

t1 = time.clock()
command = "x264 --subme 7 --progress -o client_video.264 client_file.y4m"
print command
os.system(command)
t2 = time.clock()

nopeus = t2 - t1   #Lasketaan pakkaukseen kulunut aika sekunteina

print ""
print "Pakkaaminen valmis, aikaa kului " + str(round(nopeus, 2)) + "
sekuntia."

#### </VIDEON PAKKAAMINEN LOPPUI> ####################################

#### TIEDOSTON TAKAISIN LAHETYS ######################################
tiedoston_koko = os.stat('client_video.264') #Luetaan siirrettävän tiedoston
                                             #koko
filesize = tiedoston_koko.st_size

lahetettava_filu = os.open("client_video.264",os.O_BINARY)

s.send('takaisinlahetys')    #Lähetetään tieto takaisinlähetyksen alkamisesta
s.send(str(filesize))        #Lähetetään lähetettävän tiedoston koko

vastaus = 0
vastaus = s.recv(1024)       #Vastaanotetaan palvelimen vastaus

laskin = 0                   #Käytetään statistiikan ajastukseen
data_count = 0               #Lähetetyn datan määrä
buf = 0

if vastaus == "ready":
   while 1:
      laskin += 1
      buf = os.read(lahetettava_filu, 8192)

      data_count += len(buf)

      if laskin == 4:        #Määrätään statistiikan päivitystahti
         print "L\x84hetetty",round(float(data_count) /
                float(filesize)*100,2),"\b%", data_count /
                1024,"kilotavua.","\r",   #\b backspace, \r carriage return
         laskin = 0

      if not buf:            #Suljetaan lähetys kun bufferi on tyhjentynyt
         print "L\x84hetetty", data_count / 1024, "kilotavua
                -",round(float(data_count) / float(filesize)*100,2),"\b%"
                #\b backspace

         print 'Tiedosto l\x84hetetty.\nL\x84hetetyn tiedoston koko',
                data_count, "tavua."
         os.close(lahetettava_filu)
         break

      s.send(buf)

else:
   print "Virhe tapahtunut, palvelin ei ollut valmiina vastaanottoon."
   sys.exit()
```

```
s.close()

#### </TIEDOSTON TAKAISIN LÄHETYS LOPPUI> ############################

#### VÄLIAIKAISTIEDOSTOJEN POISTO ####################################

if tempfile_poisto == True:  #Poistetaan väliaikaistiedostot

   os.remove("client_video.264")
   os.remove("client_file.y4m")

   print "V\x84liaikaistiedostot poistettu."

#### </VÄLIAIKAISTIEDOSTOJEN POISTO LOPPUI> ############################
```

LASKUESIMERKKEJÄ KAAVIOISSA KÄYTETYISTÄ LUVUISTA

Videon pituus = 100 minuuttia = 6000 sekuntia
Kehysten kokonaismäärä videossa = 6000*25 kehystä = 150000 kehystä
Jokaisen koneen pakkausnopeus = 5 kehystä sekunnissa
Käsiteltävä kehysmäärä yhdellä koneella = 150000/koneiden lukumäärä

Kahta konetta käytettäessä:
Käsiteltävä kehysmäärä yhdellä koneella = 150000/2 = 75000 kehystä
Yhden koneen nopeus = (75000/5)/60 = 250 minuuttia
(Huom. molemmat koneet pakkaavat samanaikaisesti, jolloin pakkaukseen kulunut aika yhteensä on sama kuin yhden koneen kuluttama aika.)

Verkon nopeus = 100Mbps
Tiedostokoko = 750Mt
Tiedonsiirtoon kulutettu aika, kun pakatun tiedoston kooksi arvioitiin kymmenesosa pakattavan tiedoston koosta = 750 / (100/8) + 750 / (100/8)*0,1 = 66 sekuntia = 1,1 minuuttia

Palvelin käsittelee tiedoston kahteen kertaan ennen siirto-operaatioita, sekä lopuksi yhdistää pakatut tiedostot. Yhteensä palvelimen kovalevylle tulee siis 2,1 kertainen määrä käsiteltävää tietoa alkuperäiseen tiedostokokoon verrattuna. (Huom. kovalevyn käyttöä ei huomioida tiedostoa siirrettäessä koska pullonkaulana on verkon nopeus.)

Kovalevyn nopeus = 80Mt/s
Kovalevyn toimintoihin kulunut aika = 750*2,1/80/60 = 0,33 minuuttia

Toimintoihin kulunut aika kahdella koneella ja 100Mbps verkkoyhteydellä. (Kuva 8)
Kahdella koneella: 250+1,1+0,33 = 251,43 minuuttia

Kovalevyn toimintoihin sekä tiedonsiirtoon kuluvan ajan osuus prosentteina (Kuva 15) = (tiedonsiirtoon käytetty aika + kovalevyn toimintoihin kulunut aika / kokonaisaika)*100

.y4m tiedostojen koko tavuina voidaan laskea seuraavalla tavalla: kehyksen korkeus * kehyksen leveys * 3/2 * kehyksien määrä.

100 minuutin 0,1 megapikselin YUV4MPEG2 tiedostokoko, kun FPS=10 (Kuva 18):

((0,1*1000000)*3/2*10)*60*100/1024/1024/1024=8,38Gt