This page intenionally left blank

Tallinna Pedagoogikaülikool

Informaatika Õppetool

Hajusarvutuse kasutamine õpisüsteemides IVA näitel

Bakalaureusetöö

Hillar Põldmaa

Juhendaja: Meelis Karp

Koostaja: ....." " mai 2004.a.

Juhendaja: ....." " mai 2004.a.

Osakonnajuhataja ....." " mai 2004.a.

Tallinn 2004

Sisukord

1.Sissejuhatus

IVA on TPÜ haridustehnoloogia keskuse ja informaatika osakonna ühistööna loodav veebipõhine õpihaldussüsteem, millel on mugav ja intuitiivselt õpitav eestikeelne kasutajaliides ning mille kujundamisel on kasutatud mitmeid nii arvuti- kui koolimaailmast tuttavaid metafoore. IVA on avatud lähtekoodiga vabavara, see tähendab, et igaüks võib selle tasuta oma serverisse üles seada ning vastavalt vajadusele ka lähtekoodi muuta.

Õpihaldussüsteem IVA toimib ZOPE serveri peal, mille programmeerimiseks kasutatakse DHTML-i ning Pythonit. Mõlemat võib pidada kõrgkeeleks, mis võimaldab kasutajal, kes ei ole tingimata programmeerija, kirjutada loomulikule keelele lähedasi lauseid ning mille lihtlausele vastab ülisuur arv masinakäske, samas kasutavad arvutid sisemiselt masinkoodi – ühtede ja nullide jada. Seega tuleb arvutil enne põhilise käsu täitmisele asumist täita veel üks ülesanne – transleerida kasutaja käsk arvutile arusaadvaks masinkoodiks.

Senikaua, kuni masinal on kasutajaid vähe pole selline koodi transleerimisele kuluv aeg eriliseks probleemiks, samas kui aga masinale tekib üheaegseid kasutajaid rohkem, hakkab selline koodi transleerimine halvasti mõjutama serveri käideldavust – ooteaeg päringu tegemisest vastuse saamiseni muutub väga pikaks.

Hetkeseisuga on TPÜ IVA-l umbkaudu 2000 aktiivset kasutajat, kuid see arv kasvab väga kiiresti. Samuti võib oletada, et Tallinna Ülikooli moodustumisega võetakse IVA kasutusele ka juurde tulevate koolide õpetajate ning õpilaste poolt. Hinnangute kohaselt on IVA-l 2005 aasta kevadeks umbes 10 tuhat aktiivset kasutajat, kusjuures suur osa nendest puutub IVA-ga kokku esmakordselt. Seega võib oletada, et vähemalt esialgu vormistab suurem osa kasutajatest oma masinale antavad käsud vigadega. See aga suurendab serveri koormust veelgi, kuna tal tuleb kõigepealt töödelda see vigane käsk, anda sellele mingi kasutaja jaoks arusaadav veateade ning hiljem protsessida ka korralikult vormistatud lause.

Selline kasutajatebaas seab IVA keskkonda serveerivale arvutile väga ressursimahuka ülesande. Tõenäoliselt oleks esimene lahendus sellisele probleemile suurendada masina jõudlust, kuid samas ei saa seda teha lõpmatuseni. Samuti on suure jõudlusega serverite hind mitu suurusjärku kõrgem. Samas oleks odavam lahendus kasutada tavalisi PC tüüpi arvuteid, mis oleks jõudluse tõstmiseks ühendatud omavahel paralleelselt, kuid sellega tekib kohe probleem, kuidas neid arvuteid komplekteerida ja juhtida nii, et nad omavahel koordineeritult toimiksid.

Teema edasisel uurimisel selgus, et arvutiteaduse kontekstis räägitakse arvutusmahukuse ja paralleelarvutuse juures samadest asjadest – hajusarvutusest (i.k. *distributed computing*), klastertehnoloogiast (i.k. *cluster tecnology*) ja võretehnoloogiast (i.k. *grid tecnology*). Seega on bakalaureusetöö pühendatud rohkem nende teemade uurimisele.

Käesoleva bakalaureusetöö kirjutamisel seadsin endale kolm eesmärki:

1. uurida, kuidas on mujal maailmas lahendatud hajusarvutuse printsiipidel põhinevad arvutusmahukad ülesandeid lahendavad ja/või suurt töökindlust vajavad arvutisüsteemid ning millised on selliste lahenduste head ning halvad küljed;
2. uurida teoreetiliselt kas ja kuidas on võimalik PC tüüpi arvuteid omavahel koordineeritult koos toimima saada ning kas ja kuidas on ZOPE serverit koos IVA-ga võimalik taolistel süsteemidel käivitada;
3. võimaluse korral panna üks selline süsteem ka reaalselt kokku ning testida seda ZOPE ja IVA-ga töömahukate protsesside käivitamisel.

Käesoleva bakalaureusetöö eesmärgiks on uurida erinevaid hajusarvutusel põhinevaid tehnoloogiaid ning võimalusel kirjutada eestikeelne abimaterjal hajusarvutuse tehnoloogiatel põhineva serveri ehitamiseks ja konfigureerimiseks. Samuti on käesoleva töö eesmärgiks välja pakkuda üks võimalikest lahendustest õpihaldussüsteemi IVA jaoks kiire ning töökindla, kuid samas ka hinnalt vastuvõetava serveri ehitamiseks.

2. Hajusarvutuse mõiste

Järgnevalt püütakse anda ülevaadet hajusarvutuse mõistest, hajusarvutuse eriliikidest, ajaloost ning tänapäevast.

2.1. Definitsioon

Hajusarvutus (jaotatud andmetöötlus) on arvutisüsteem, kus mitu omavahel ühendatud arvutit jagavad ühist, süsteemile täitmiseks antud, ülesannet[1].

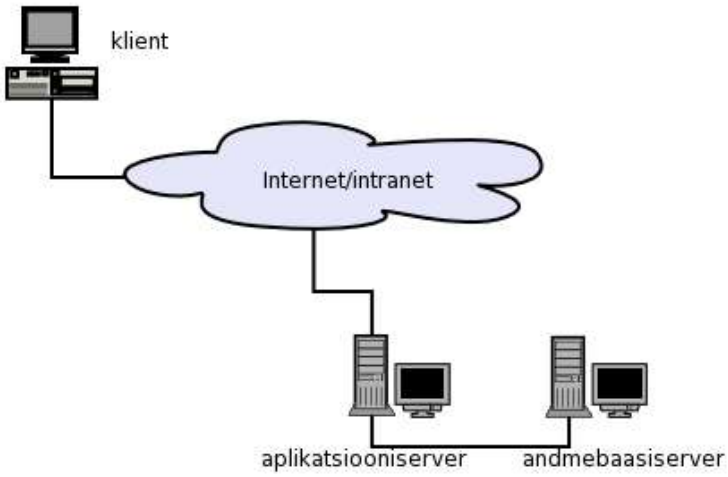
Hajutatud süsteem tähendab arvutisüsteemi ressursside hajutamist eri asukohtade vahel. Samal ajal sõna "süsteem" tähendab, et nende jagatud ressursside baasil peab moodustuma terviklik ja koostöötav arvutilahendus. Ressurssideks nimetatakse seejuures riist- ja tarkvarakomponente, samuti ühiskasutatavaid andmeid. Tekstitöötluse juures võib näiteks tekstiredaktor olla ühes masinas, õigekirja kontrollija (*speller*) teises masinas ja sünonüümisõnastik (*tesaurus*) kolmandas masinas. Kusjuures mõnedes hajusarvutus-süsteemides võib iga arvuti olla erineva operatsioonisüsteemiga.

2.2. Hajusarvutuse erinevad liigid

Hajutatud ressursside baasil sellise süsteemi loomine, mis näeb kasutajate jaoks välja nii, nagu oleks tegemist ühe tsentraalse arvutisüsteemiga, on väga ahvatlev mõte. Selle teeb eriti ahvatlevaks lootus kanda tsentraliseeritud süsteemide head omadused üle hajutatud ressursside kogumile, samaaegselt vähendades tsentraliseeritud süsteemide negatiivseid omadusi. Järgnevalt vaatleme erinevaid võimalusi arvutite omavaheliseks koostööks.

2.2.1. Klient-server

Äriettevõtetes on laialdaselt kasutusel hajusarvutuse üks eriliik, mida nimetatakse klient-server tehnoloogiaks. Andmetöötluse jaoks vajalikud erinevad osad paigutatakse erinevatesse võrgus olevatesse arvutitesse, kusjuures iga arvuti on optimeeritud täitma just talle usaldatud ülesannet. Tüüpilises süsteemis, kus kasutatakse andmetöötluse kolmekihilist mudelit, on kasutajaliides kasutaja juures olevas arvutis, ärirakendus toimib serveris ning andmebaasirakendus mingis kolmandas masinas, mis serveerib andmeid mitmele erinevale ärirakendusele. Üldjuhul kasutab sealiiki jaotatud andmetöötlus mingit klient-server tüüpi kommunikatsioonimudelit. Tänapäevaks on tekkinud ka teenusepakkujad, kes pakuvad üle Interneti mõnda ärirakenduse juures vajalikku komponenti, näiteks andmebaasiserverit. Tüüpilise klient-server rakenduse näide on joonisel 1.



Joonis 1: Klient-server rakendus

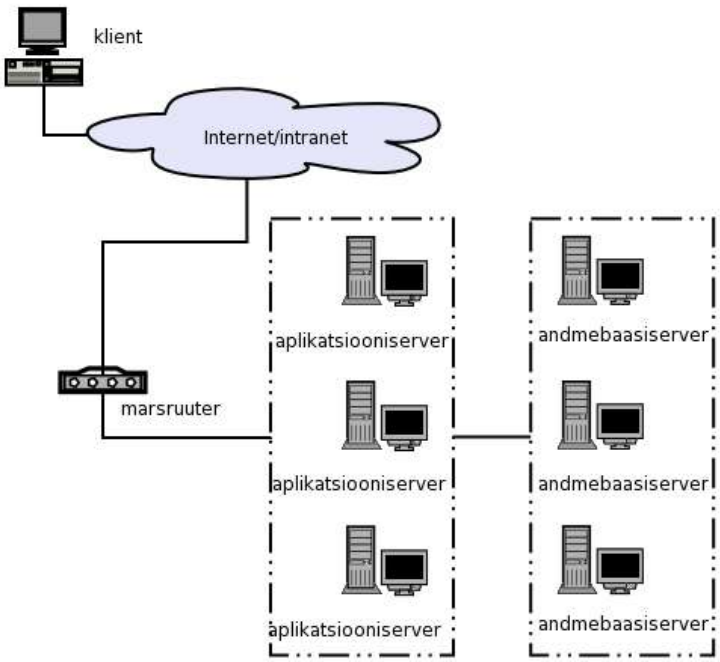
2.2.1.1.Näiteid

Sellise meetodi näiteid on väga palju. Kõige tuntumaks võiks vast pidada Oracle[2] poolt pakutavaid lahendusi. Samas on selliseid rakendusi ka suhteliselt lihtne ise ehitada. Tüüpilisemateks juhtudeks on veebiserveri ja andmebaasi kooslus, kus veebiserver on ühes masinas ning andmebaas teises.

2.2.2.Klaster

Klient-server tehnoloogia edasiarenduseks võib pidada süsteemi, kus serveri jõudluse suurendamise huvides on paralleelselt ühendatud mitu aplikatsiooniserverit ja/või andmebaasiserverit.

Joonisel 2 on kujutatud sellise klastri põhimõttelist skeemi, täpsustamata marsruuteri ja klastris olevate serverite omavahelisi ühendusi. Juhul, kui klastri oluliseks nõudeks on pidev kättesaadavus, siis paigaldatakse marsruuteriga paralleelselt ka teine, tagavara marsruuter, mille seadistused on põhimarsruuteriga identsed ning mis põhimarsruuteri tõrke korral võtab tema ülesanded üle. Sellist süsteemi nimetatakse vea korral asendavaks klastriks (i.k. *Simple Failower cluster*).



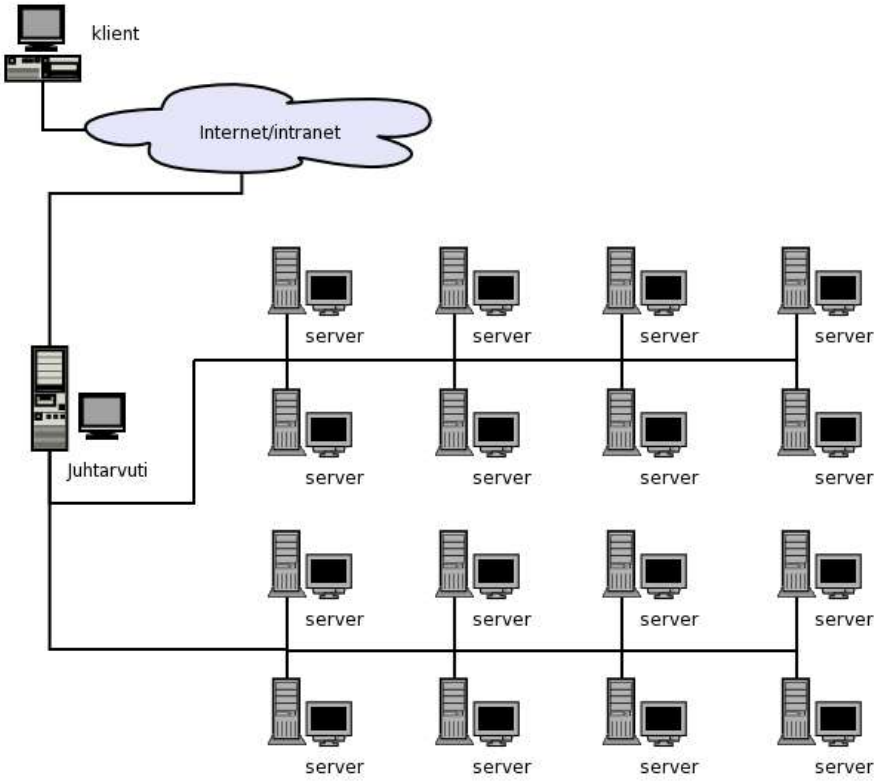
Joonis 2: Tüüpiline klaster

2.2.2.1.Näiteid

Kõige tuntumaks klastersüsteemi näiteks võib vast pidada Google otsingumootorit. Googlel on üle maailma laiali sadu aplikatsiooniserveri-andmebaasiserveri komplekte, kusjuures igas aplikatsiooniserveris toimib otsimootori rakendustarkvara. Päringu saabudes analüüsib marsruuter selle päringu lähteaadressi ning suunab ta siis töötlemiseks kliendile kõige lähemal olevale aplikatsiooniserverile. Aplikatsiooniserver töötleb päringut ning saadab töödeldud päringu edasi andmebaasiserverile. Viimane annab vastuse aplikatsiooniserverile, aplikatsiooniserver töötleb vastuse inimesele vastuvõetavale kujule ning saadab selle siis otse kliendile. Eelpool toodud näide Google kohta on ülimalt utreeritud, kuna siin on vaatluse alt välja jäetud andmebaasi koostamine ning tarkvara ja andmebaasi sünkroniseerimine erinevate sõlmede vahel.

2.2.3.Võre

Kõige tihedamini kasutatakse võre ehk *gridi* mõistet tähistamaks mõnda suurt koostöövõrku, kus asub palju iseseisvaid, kuid loogiliselt ühiseks tervikuks ühendatud servereid ja/või klastreid.



Joonis 3: Tüüpiline võre

Siinjuures ei ole oluline kuskohas arvutid füüsiliselt asuvad. Võresid kasutatakse eriti töömahukate ülesannete korral ning sellise süsteemi üheks esimeseks kasutusalaks oli krüpteeritud sõnumite murdmine grupi poolt, mida tänapäeval tuntakse distributed.net nime all. Tüüpilises võre näide on joonisel 3.

2.2.3.1.Näiteid

Kõige tuntumaks sellise võrgu näiteks on ilmselt SETI@home nimeline projekt, mis sai alguse Berkeley Ülikooli juures juba 1983 aastal, kuid mis käivitus täismahus alles 1997.

SETi@home põhineb ideel, et maailmas on tuhandeid arvuteid, mis suure osa ajast on küll sisse lülitatud, kuid seisavad jõude – jooksutavad lihtsalt ekraanipimendajat (i.k. screensaver). Selle asemel võiksid nad jooksutada mõnda teist ekraanipimendajat, mis ühtlasi ka midagi kasulikku teeks. Küsitluste teel tehti kindlaks, maailmas on küllaldaselt palju personaalarvuteid ja millede omanikud lubavad neid arvuteid kasutada mingi suure ja/või keeruka probleemi lahendamiseks.

Projekti põhiliseks osaks on raadioteleskoop, mis püüab kosmosest saabuvaid signaale. Nendest signaalidest püütakse välja filtreerida mõistuslikule elule viitavaid osiseid, mis on ülimalt töömahukas andmete analüüs. Raadioteleskoobi juures asuvast arvutist saadetakse

kosmosesignaalid edasi Berkeley Ülikooli juures asuvasse keskserverisse, mis jagab ülesande väiksemateks tükkideks. Klientmasinasse installeeritud programm võtab, jõudemomendi tekkides, ühendust Ülikooli keskserveriga ning tõmbab endale tükikese arvutusülesandest. Senikaua, kuni masinal on jõudeaeg, tegeleb ta selle kõrvalülesandega – andmete analüüsiga, kuid siis kui kasutajal on vajalik arvutit kasutada, pannakse andmeanalüüs seisma ning tegeletakse kasutaja poolt antud käskude täitmisega. Siis kui andmed on analüüsitud, saadetakse tulemused ülikooli serverisse tagasi ning tõmmatakse sealt uus jupp ülesannet.

Kuna kasutatavate masinate hulk on kümnetes tuhandetes, siis teeb üks arvuti sellest töömahukast ülesandest väga väikese osa ning seda pole klientmasinas peaaegu märgata, samas kui lõpptulemusena on läbi uuritud tohutus koguses informatsiooni.

Tasub tähele panna põhimõttelist võimalust kasutada mingit arvutit ülesannete täitmiseks klastri või võre koosseisus, kuid samas täidab see arvuti muid ülesandeid iseseisva serveri või tööjaamana.

2.3.Hajusarvutussüsteemidel toimivad rakendused

Hajussüsteemi eesmärgiks on hulga autonoomsete arvutite baasil tervikliku süsteemi loomine nii, et keerukas seadmete ja ressursside struktuur jääks süsteemi kasutajatele nähtamatuks (i.k. transparent). Kuna hajussüsteem võib koosneda väga erinevast riist- ja tarkvarast, siis on süsteemi omaduste tagamisel oluline, et rakendus toetaks paralleelselt mitmel arvutil toimimist. Seega on oluline, et erinevates rakendustes olevate objektide omavahelise suhtlemise meetodid oleks standardiseeritud. Käesoleval ajal on kaks enamkasutatavat hajusarvutussüsteemidel toimivate rakenduste standardit – CORBA (*Common Object Request Broker Architecture*) ja DCOM (*Distributed Component Object Model*).

CORBA[4] on OMG (*Object Management Group*)[5] poolt arendatav komplekt standardeid, millest osa on heaks kiidetud ka ISO (*International Standardisation Organisation*) poolt. Standardite komplekt põhineb omakorda järgnevatel standarditel:

- Common Warehouse Metamodel (CWM);
- Meta-Object Facility (MOF);
- Unified Modeling Language (UML);
- Extensible Markup Language (XML);
- XML Meta-Data Interchange (XMI).

CORBA on põhjaks ka programmeerimiskeeles Javas, täpsemalt Java-RMI's (*Remote Method Invocation*) kirjutatud hajusrakendustele.

DCOM põhineb Avatud Tarkvara Fondi (*Open Software Foundation*) DCE-RPC spetsifikatsioonil, mida on Microsofti poolt edasiarendatud nii, et ta sobiks ActiveX protokolliga. CCOM kohta leiab tarkvara arendamiseks vajalikke materjale Microsoft TecNet'ist[6].

Lisaks neile kahele on standardeid veelgi. Eelpool oli juttu DCE-RPC spetsifikatsioonist. DCE on lühend sõnaühendist *Distributed Computing Environment* ning RPC lühend sõnaühendist *Remote Procedure Calls*. Neid kasutatakse tänapäeval palju suurte ärirakenduste alusena. Lisaks sellele on mõnedes teadusvõrkudes suurt mälumahtu ja/või arvutusvõimsust vajavate simulatsioonirakenduste programmeerimisel aluseks PVM (*Parallel Virtual Machine*), MPI (*Message Passing Interface*), LAM-MPI (*Local Area Multicomputer*) nimelised standardid.

Kuna hajusarvutus on plahvatuslikult kasvav ala, siis ei pruugi eelnev olla ammendav loetelu standarditest. Samuti ei jõua keegi loendada hajusarvutussüsteemidele kirjutatud konkreetseid rakendusi, kuid selliste rakenduste ühisteks iseloomujoonteks on alati jõudlus, skaleeritavus, töökindlus ning suur käideldavus.

2.4.Mõisted hajusarvutusvõrkude kontekstis

Hajusarvutusvõrkude kontekstis on kasutusel järgnevad mõisted:

- Virtuaalserver (i.k. *Virtual server*) – arvutite koostöövõrk, mis kasutajale paistab ühe serverina.
- Virtuaalaadress – hajusarvutussüsteemi (virtuaalserveri) väline IP-aadress.
- Reaalserver (i.k. *Real server*) – virtuaalserveris olev reaalne server.
- Direktor – virtuaalserverit juhtiv arvuti. Vahel on kirjanduses kasutatud ka mõistet peasõlm (*master node*).
- Serveri sõlm (i.k. *server node*) – suvaline virtuaalserveri koosseisu kuuluv seade (reaalserver, marsruuter jne).

2.5. Virtuaalserverite tarkvara

2.5.1. Linuxi Virtuaalserveri projekt

Linux Virtuaalserver Project[7] on IP-l (*Internet Protocol*) põhinev tarkvara komplekt virtuaalserveri ehitamiseks. Tarkvarakomplekt koosneb põhiliselt Linuxi kerneli tuumast, mida on modifitseeritud nii, et võimaldaks pakettide juhtimist OSI-mudeli neljandas kihis. Samuti on sellesse modifikatsiooni sisse ehitatud erinevad pakettide ümbersuunamise meetodid ning erinevad koormuse jagamise algoritmid. Pakettide ümbersuunamisest ning koormusjagamise algoritmidest tuleb käesolevas töös juttu edaspidi.

Lisaks kerneli modifikatsioonile on Linux Virtuaalserver projekti raames välja arendatud hulk utiliite, mis võimaldavad jälgida reaalserverite seisundit, saata serveri administraatorile teateid serverite oleku kohta ning dünaamiliselt reageerida tegelikule olukorrale – päringuid erinevate reaalserverite vahel ümber suunata, järjekorda seada või juhul, kui kõik reaalserverid on töövõimetud, siis näidata mingit kasutajale arusaadavat veateadet.

Enamus maailmas toimivatest akadeemilistest virtuaalserveritest põhineb selle projekti raames välja töötatud tarkvara komponentidel.

2.5.2. Oracle

Ärimaailmas on suurimaks hajusarvutustehnoloogia lahenduste pakkujaks Oracle. Põhiliselt on tema poolt pakutavad lahendused klient-server tehnoloogiate edasiarendused, millele on lisatud võimalused mitme aplikatsiooniserveri ja/või andmebaasiserveri paralleelseks kasutamiseks aga samuti ka vahendid reaalserverite jälgimiseks ning vajadusel virtuaalserveri automaatseks ümberkonfigureerimiseks ja/või serveri administraatorile teate saatmiseks. Tänapäeval on Oracle võtnud kursi vabavaraliste lahenduste suunas ning pakub oma tooteid koos SuseLinux ja Red Hat Enterprise Linux tarkavarapakettidega [8].

2.5.3.Cisco

Joonis 4: Cisco OSI 4-kihi marsruuter

Ka marsruuterite, jaoturite ning kommutaatorite suurvalmistaja Cisco pakub oma valikus virtuaalserveri ülesseadmiseks sobivat OSI-mudeli neljanda kihi kommutaatorit. Lisaks on selles komplektis ka vajalik tarkvara reaalserverite jälgimiseks ning vajadusel serverite administraatorile teadete saatmiseks aga samuti ka palju teisi vajalikke utiliite[9]. Joonisel 4 on kujutatud üks OSI 4-kihi marsruuter.

2.5.4.Microsoft

Microsoft pakub oma lahendusi põhiliselt klient-server tehnoloogiate kontekstis. Põhilised märksõnad oleksid ActiveX ja ASP.NET. Samuti pakub Microsoft *Simple Failover* tehnoloogiat[5].

2.5.5.Teisi valmislahendusi

Kuna ülikoolide ja teadusasutuste põhiliseks probleemiks on finantsvahendite puudus, kuid samas on olemas tugev kompetentsi tase, siis sobiksid akadeemilisse maailma ilmselt kõige paremini vabavaralised lahendused, kuna nende lähtekood on avalik. Samas on erinevaid Linuxi distribuutoreid, kes pakuvad mingisugust virtuaalserveri tarkvara ning kellelt on võimalik saada ka tasulist kasutaja- ja/või tehnilist tuge, piisavalt palju. Enamasti põhinevad need valmislahendused Linux Virtuaalserver projekti raames välja töötatud tarkvaral, seega on juhul, kui distribuutor peaks mingil põhjusel oma toe lõpetama, on piisavalt lihtne leida uus kasutaja- ja/või tehnilise toe pakkuja. Samuti on sellisel juhul võimalus oma virtuaalserver lihtsate vahenditega migreerida mõnele teisele Linuxi distributsioonile.

Järgnevalt katsutakse anda ülevaade mõnedest Linuxi distribuutoritest, kes pakuvad valmislahendusi virtuaalserverite ehitamiseks, koos pakutava tarkvara lühikirjeldusega.

Nimekiri ei ole täielik, samuti on tema järjekord juhuslik.

Knoppix – pakub ClusterKnoppix nimelist paketti[10]. See, nagu ka Knoppix ise, põhineb Debiani distributsioonil ning sisaldab kõiki virtuaalserveri ehitamiseks ning konfigureerimiseks vajalikke komponente. Pakutakse teda alglaadiva CD-na, kuid samas on võimalus ta ka kõvakettale installeerida. Plaadi tõmmised on vabalt allalaetavad, kuid tugi on tasuline.

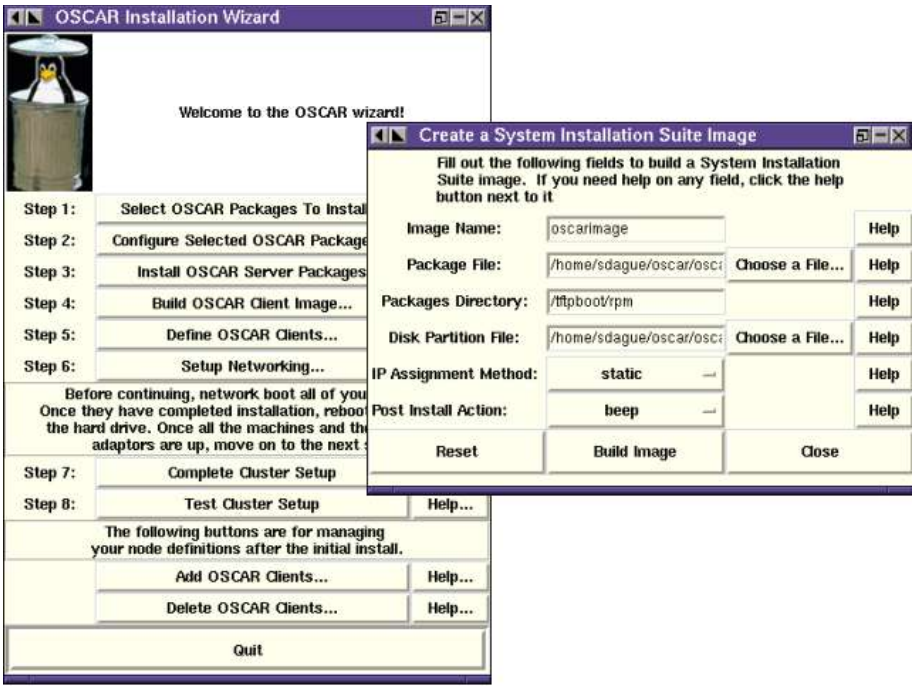
MandrakeLinux – pakub CLIC nimelist paketti[11], mis sisaldab vastavalt modifitseeritud kernelit ning vajalikke konfigureerimis- ja jälgimisutiliite. CLIC nõuab kõvakettale installeerimist ning installeerimis CD tõmmis on vabalt allalaetav. Samuti, nagu Knoppix'il, on tema tugi tasuline.

Red Hat – pakub Red Hat Enterprise Linux nimelist paketti[12], mis on mõeldud rohkem ärilahendustes kasutamiseks. See sisaldab kõiki virtuaalserveri ülesseadmiseks vajalikke komponente. Lisaks virtuaalserverile sisaldab ta ka palju teisi ärimaailmas vajalikke rakendusi. Selle installeerimiseks vajalikud komponendid ning samuti tugi on tasulised, kuid lähtekood on allatõmmatav tasuta.

2.5.6. Eriprogrammid

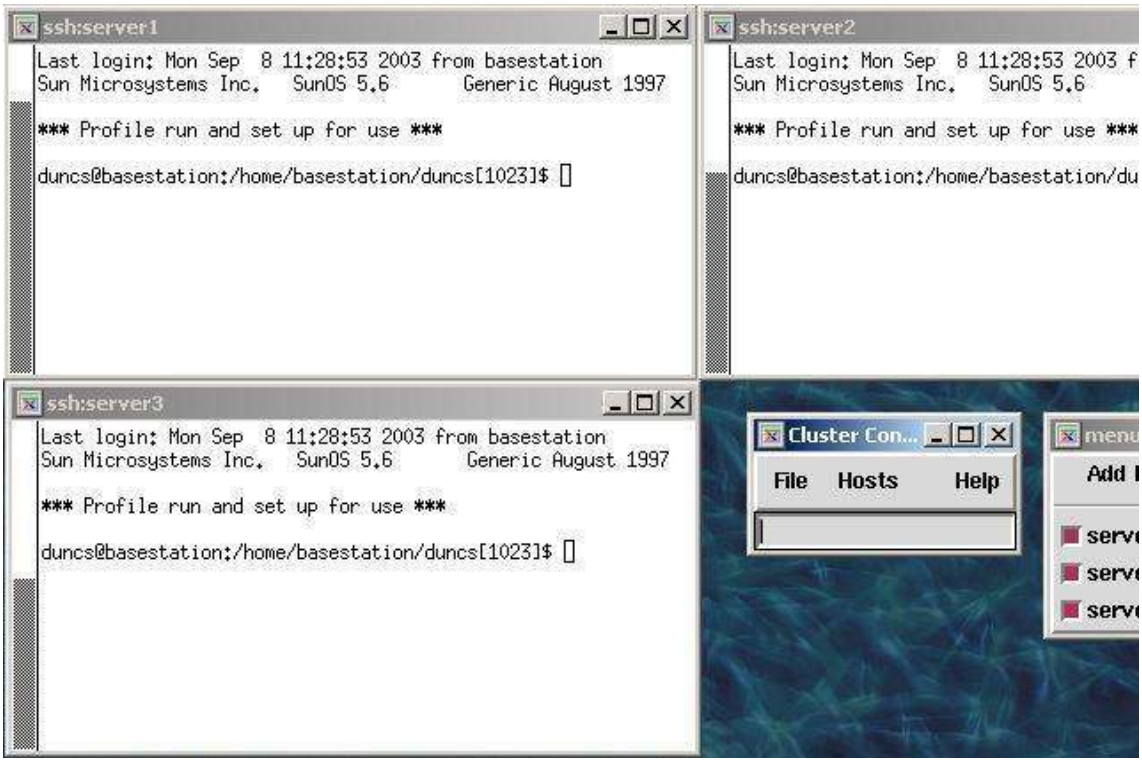
Järgnevalt püütakse anda ülevaadet klastrite ja võrede ülesseadmiseks ja haldamiseks mõeldud lisatarkvarast. Alljärgnev loetelu ei ole sugugi ammendav, kuna täieliku nimekirja koostamine kogu selleks otstarbeks oleva tarkvara kohta on üsna raske, sest neid on suhteliselt palju ning see nimekiri muutub pidevalt. Selle töö autor on võimalikust nimekirjast teinud valiku lähtuvalt programmi litsentsitingimustest, kusjuures eelistatud olid GNU-GPL või sellele lähedaste tingimustega programmid, ning arendamise aktiivsusest. Käesolevast peatükist on välja jäetud ka ülevaade selle töö raames kasutatud programmist Keepalived, kuna sellest tuleb pikemalt juttu käesoleva töö praktilises osas.

OSCAR (Open Source Cluster Application Resource)[13] on GNU-GPL all välja antud programm, mille autoriks on Sean Dague. Programm pakub kõiki utiliite mis on vajalikud klastri ülesseadmiseks, administreerimiseks ja jälgimiseks. Programmi arendamist alustati 2001. aastal ning hetkel on viimane stabiilne väljalase versiooninumbriga 2.3.1. Programmi töökeskkonnaks on X11. OSCARi installeerimispaketid (*rpm*-id) on saadaval Mandrake, Red Hat-i ja Debian-i distributsioonide jaoks. Joonisel 5 on kujutatud programmi tööakna ekraanipilt.



Joonis 5: Programmi OSCAR ekraanipilt

Cluster SSH[14] autoriks on Duncs. Programmi arendamist alustati 2002. aastal ning hetkel on jõutud versioonini 1.56. Programmil on graafiline kasutajaliides, mis toimib X11 keskkonnas. Erisuseks on see, et ühendustes kasutatakse SSH protokolli. Litsentsitingimusteks on GNU-GPL. Joonisel 6 on programmi tööakna ekraanipilt.



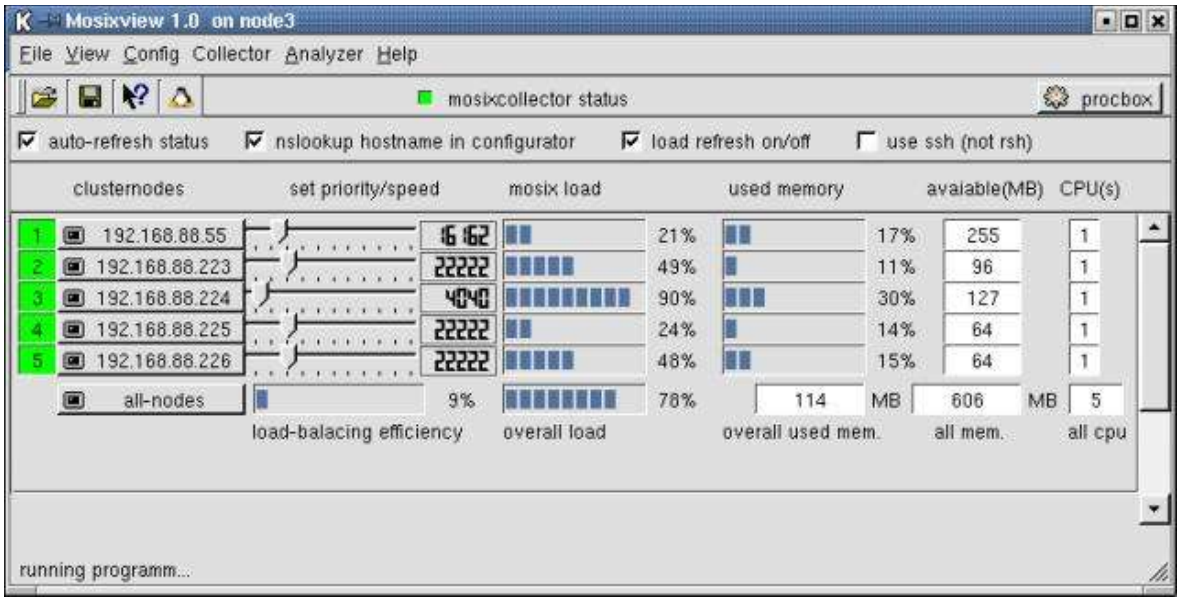
Joonis 6: Programmi Cluster SSH ekraanipilt

Kimberlite Webmin moodul[15] on veebiserver Apache konfigureerimisutiliidi Webmin lisamoodul. Programmi autoriks on IDEALX, kes on programmi arendamist alustanud 2002. aastal ning käesolevaks ajaks jõudnud versioonini 1.0. Nagu juba öeldud, on tegemist Webmini lisamooduliga, seega toimub programmi konfigureerimine üle veebi brauseri vahendusel. Litsentsitingimused vastavad GNU-GPL toodutele. Kimberlite Webmin mooduli ekraanipilt on toodud joonisel 7.

Joonis 7: Kimberlite Webmin mooduli ekraanipilt

OpenMosix Cluster for Linux[16] ja tema graafiline kasutajaliides **openMosixview**[17]**.** Neist esimese autoriks on Moshe Bar ning graafilise liidese autoriks M. Rechenburg. Siinjuures on vajalik tähele panna, et tegelikuks tööks on mõeldud neist esimene ning teine on ainult graafiline liides. Programmi esimene versioon valmis 2001. aasta alguses ning tema graafiline liides umbes pool aastat hiljem. Hetkel on programmi arendamisel jõutud versioonini 2.4.22-3. Graafiline liides toimib X11 keskkonnas, samuti on olemas tema versioonid populaarsematele Linux-i töölauakeskkondadele – KDE-le ja Gnome-le. OpenMosixview tööakna ekraanipilt on kujutatud joonisel 8.

Ultra Monkey[18] autoriks on Simon Horman. Tegemist on ainult käsurealt



Joonis 8: Programmi openMosixview ekraanipilt

konfigureeritava, kuid samas täiskomplektse paketiga virtuaalserveri loomiseks, administreerimiseks ning jälgimises mõeldud tarkvarast. Hetkel on programmi arendamisel jõutud versioonini 2.0.1. Programm on välja antud GNU-GPL litsentsi all, ning temast on installatsioonipaketid (*rpm*-id) nii Debianile, Fedora Corele, Red Hat'ile kui ka Red Hat Enterprise Linux'ile.

Nagu juba öeldud ei olnud see ammendav loetelu kõikidest selleks otstarbeks ettenähtud programmidest. Samuti on käesolevas töös täielikult vaatluse alt välja jäetud klastritel ja võredel kasutatavad rakendusprogrammid.

2.6.Näiteid toimivatest süsteemidest

Maailmas on tuhandeid hajusarvutuse mingil alamhulgal põhinevaid servereid[19]. Enamasti on need mingil klient-server tehnoloogial põhinevad süsteemid. Samuti on maailmas sadu võretehnoloogiat kasutavaid servereid. Enamasti kuuluvad nad mõne suurema ülikooli juurde ning on mõeldud mingi ülimalt arvutusmahuka ülesande lahendamiseks. Suure jõudlusega arvutite kohta peab arvet veebileht aadressil http://www.top500.org/. Samas tahaks eriti esile tuua kahte võretehnoloogiat kasutavat virtuaalserverit.

2.6.1.NorduGrid

NorduGrid[20] on võretehnoloogia uurimise ning arendamisega tegelev koostööprojekt, mille eesmärgiks on avatud lähtekoodiga võresüsteemide tarkvara arendamine, juurutamine ning kasutajatoe tagamine. NorduGrid projekt sai alguse 2002. aastal. Käesoleval ajal pakub NorduGrid koostööprojekt virtuaalserveri ehitamiseks vajalike tarkvarapakketide täiskomplektset lahendust, mis on lihtne, skaleeritav ning lihtsalt migreeritav.

Nagu juba nimetatud, on NorduGrid koostööprojekt. Selle raames teevad koostööd mitmed võretehnoloogia arendamisega tegelevad organisatsioonid, näiteks SWEGRID, Dansk Center for Grid Computing ja Nordic Data Grid Facility.

NorduGrid ise teeb tarkvaraarenduse eesmärgil koostööd näiteks EU DataGrid ja LHC Computing Grid projektiga. NorduGrid osaleb samuti Põhja-Euroopa Võresüsteemide Konsortsiumis (North European Grid Consortium).

2.6.2.Eesti Grid

Eestis on võretehnoloogia uurimisega suhteliselt vähe tegeletud. Koostööprogramm, eesmärgiga luua Eestisse teadus- ja arendustegevuse arenguks vajalik superarvutuskeskus, sai alguse alles 2003 aastal ning selle programmi raames hangitud saabusid esimesed reaalsed arvutid veebruaris 2004[21].

Eesti GRID juhtivaks arendajaks on haridus- ja teadusvõrke haldav EENet, arengu esimeses etapis püüab luua baasinfrastruktuuri mis kataks minimaalsed arvuti- ja inimressursi vajadused kohaliku arengu käivitamiseks.

Eesti GRID organisatsiooniline struktuur on avatud kõikidele teadus- ja arenduskeskustele, sealhulgas ka ettevõtetele.

Eestis on sellise tehnoloogia kasutuselevõtt ja arendamine, ennekõike inimressursi ja know-how mõttes, vajalik eelkõige sellepärast, et võimaldada meie teadusharudel püsida konkurentsivõimelisena. Samuti on see eelduseks uute arvutuslike teadusvaldkondade tekkele. Eesti GRID osaleb ka rahvusvahelistes projektides, mistõttu on meie teadlastel lisaks oma arvutiressurssidele ligipääs ka teiste riikide arvutuskeskustele.

2.7. Virtuaalserveri ehitus

2.7.1. Riistvara

Kuigi riistvara seisukohalt ei ole virtuaalserver otseselt eriti nõudlik, kasutatakse virtuaalserverites siiski vägagi võimsaid arvuteid. Seda eelkõige sellepärast, et iga virtuaalserverile saadetud päringu töötlemise aeg sõltub eelkõige just reaalserverite jõudlusest. Samuti sõltub virtuaalserveri töökiirus suurte andmemahtude liigutamise korral reaalserverite vahelise ühenduse kiirusest. Kiire ühenduse tekitamiseks lähestikku asuvate arvutite vahel kasutatakse *FireWire*'l või *GigaBit Ethernetil* põhinevaid lahendusi. Eelpool toodust järeldub, et konkreetse lahenduse jaoks vajalik riistvara konfiguratsioon tuleb valida spetsiaalselt ja et see sõltub reaalserverite arvust, nende asukohast üksteise suhtes, kasutatavast rakendusest ning virtuaalserveri oodatavast koormusest.

2.7.2. Reaalserverite vahelised ühendused

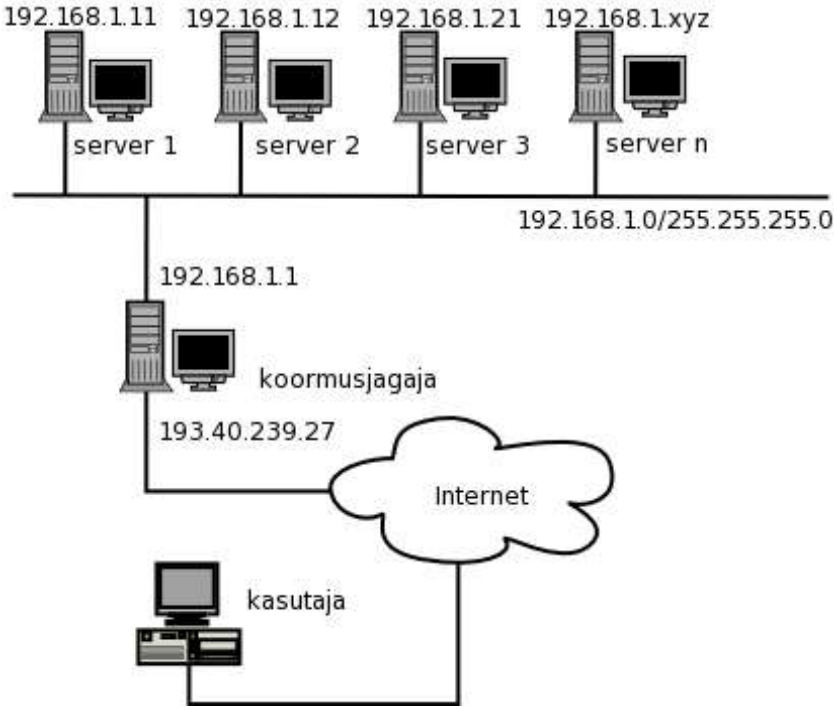
2.7.2.1. NAT-iga virtuaalserver

IPv4 aadressruumi mahuliste piirangute tõttu aga samuti ka mõnedel turvalisuse kaalutlustel kasutatakse virtuaalserveri sisevõrgus spetsiaalseid sisevõrkude jaoks ettenähtud aadressivahemikke (nagu näiteks 10.0.0.0/255.0.0.0, 172.16.0.0/255.240.0.0 või 192.168.0.0/255.255.0.0). Samas ei saa neid IP-aadresse kasutada avalikus Internetis. Seega tekib vajadus NAT (*Network Adress Translation*) järele siis, kui sisevõrgus olev arvuti soovib suhelda välisvõrgus olevate masinatega.

Kui aadressi teisendatakse N-N, siis kutsutakse seda staatiliseks võrguaadressi teisenduseks. Kui aadressi teisendus toimub M-N, kusjuures M on suurem kui N, siis kutsutakse seda dünaamiliseks võrguaadressi teisenduseks. Võrguaadressi värati (i.k. *port*) suunamine, on tavalise NAT-i laiendus. Sel viisil on võimalik teisendada mitmed sisevõrgu aadressid ning nende TCP/UDP väratid üheks välisvõrgu aadressiks ning selle TCP/UDP väratiteks. Sellise N-1 meetodiga on teostatud IP-aadressi maskeraad ning seda kasutatakse enamasti ka virtuaalserverites reaalserverite varjamiseks. NAT-i kohta annab põhjalikumat informatsiooni Internet Engineering Task Force (IETF) RFC-st (*Request for Comments*) number 1631[22].

2.7.2.2.Kuidas toimib virtuaalserver NAT-i kaudu

Kui kasutaja saadab päringu (TCP paketi) virtuaalserveri teenuse suunas, siis saadab ta päringu virtuaalserveri IP peale (koormusjagaja väline IP-aadress) ning see võetakse koormusjagaja poolt vastu. Koormusjagaja uurib selle paketi sihtaadressi ja väratit. Juhul, kui need sobivad virtuaalserveri poolt osutatavate teenuste tabeliga, siis valitakse vastavalt mõnele edaspidi kirjeldatud koormusjagamise algoritmile mõni virtuaalserveris olev reaalserver ning ühendustetabelisse lisatakse algatatud ühenduse kohta kirje. Seejärel kirjutatakse päringu paketi sihtaadress üle valitud serveri aadressiga ning päring suunatakse edasi reaalserverile. Kui sissetulev pakett kuulub ühenduse juurde ning valitud server on olemas, siis kirjutatakse tema sihtaadress koormusjagaja poolt ringi ning suunatakse



Joonis 9: NAT põhimõtteline skeem

päringut teenindavale serverile. Kui päringule saabub vastus, siis kirjutab koormusjagaja lähteaadressi üle virtuaalserveri omadele vastavate väärtustega. Kui ühendus lõpetatakse sihilikult või ajalimiidi ületamise (i.k. *timeout*) tõttu, siis kustutatakse vastav kirje ka ühendustetabelist. NAT ühenduse põhimõtteline skeem on kujutatud joonisel 9.

Reaalserverid võivad toimida täiesti suvalise operatsioonisüsteemiga, ainukeseks tingimuseks on see, et operatsioonisüsteem peab toetama TCP/IP-d ning vaikelüüsiks peab olema koormusjagaja (selles näites 192.168.1.1)

Nii näiteks: Kogu liiklus aadressile 193.40.239.27 väratisse 80 suunatakse, koormuse

jagamise algoritmi arvestades, reaalserveri aadressidele 192.168.1.11 porti 8080 ja 192.168.1.12 porti 9080. Liiklus, mis suundub aadressi 193.40.239.27 väratisse 21, suunatakse ümber reaalserveri IP-aadressile 192.168.1.12 porti 9021 (vt tabel 1).

protokoll	virtuaalne IP aadress	värat	reaalne IP aadress	värat	kaal
TCP	193.40.239.27	80	192.168.1.11	8080	1
			192.168.1.12	9080	2
TCP	193.40.239.27	21	192.168.1.12	9021	1

Tabel 1: Suunamised NAT võrgus

Pakettide ümberkirjutamise reeglid on järgnevad:

Sissetuleva päringu pakett omab lähteaadressi ja sihtaadressi näiteks niimoodi:

Lähteaadress 82.131.28.246:3456 Sihtaadress 193.40.239.27:80

Oletame, et koormusjagaja valib reaalserveriks 192.168.1.12:9080. Sel juhul kirjutatakse aadressid ümber niimoodi:

Lähteaadress 82.131.28.246:3456 Sihtaadress 192.168.1.12:8080

Vastus koormusjagajale näeb välja niimoodi:

Lähteaadress 192.168.1.12:8080 Sihtaadress 82.131.28.246:3456

Päringu pakettide aadressid kirjutatakse taas ringi ning saadetakse kliendile nii:

Lähteaadress 193.40.239.27:80 Sihtaadress 82.131.28.246:3456

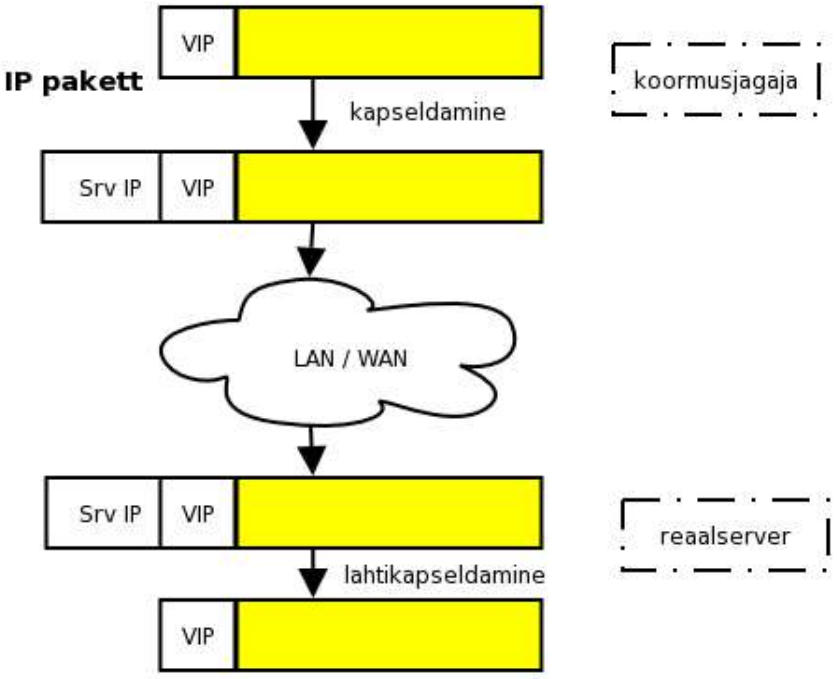
2.7.2.3.IP-tunneliga virtuaalserver

IP-tunneldamine (i.k. *IP-tunneling*) mida nimetatakse ka IP-kapseldamine (i.k. *IP-encapsulation*) on meetod, kus IP-datagramm kapseldatakse teise IP-datagrammi sisse mistõttu tekib võimalus ühele IP-aadressile tulevaid päringuid suunata ümber teisele IP-aadressile. IP-kapseldamist kasutatakse sageli laivõrkudes, mobiilvõrkudes ja multisaate (i.k. *multicast*) võrkudes. IP-tunneldamist käsitleb põhjalikumalt IETF RFC1853[23].

2.7.2.4.Kuidas toimib IP-tunneliga virtuaalserver

Kui kasutaja pöördub virtuaalserveri poolt osutatava teenuse poole, saabub virtuaalserverile pakett. Koormusjagaja uurib paketi sihtaadressi ning väratit. Juhul, kui need sobivad virtuaalserveri poolt osutatava teenusega, valitakse vastavalt jagamise algoritmile välja mõni reaalserver ning ühenduse alustamise kohta lisatakse kirje ühenduste tabelisse. Seejärel kapseldab koormusjagaja paketi IP-datagrammi ning saadab ta edasi valitud serverile. Kui mõni edaspidi sissetulev pakett kuulub selle ühenduse juurde ja valitud server on ühenduste tabelist leitav, kapseldatakse sissetulev paket ja suunatakse

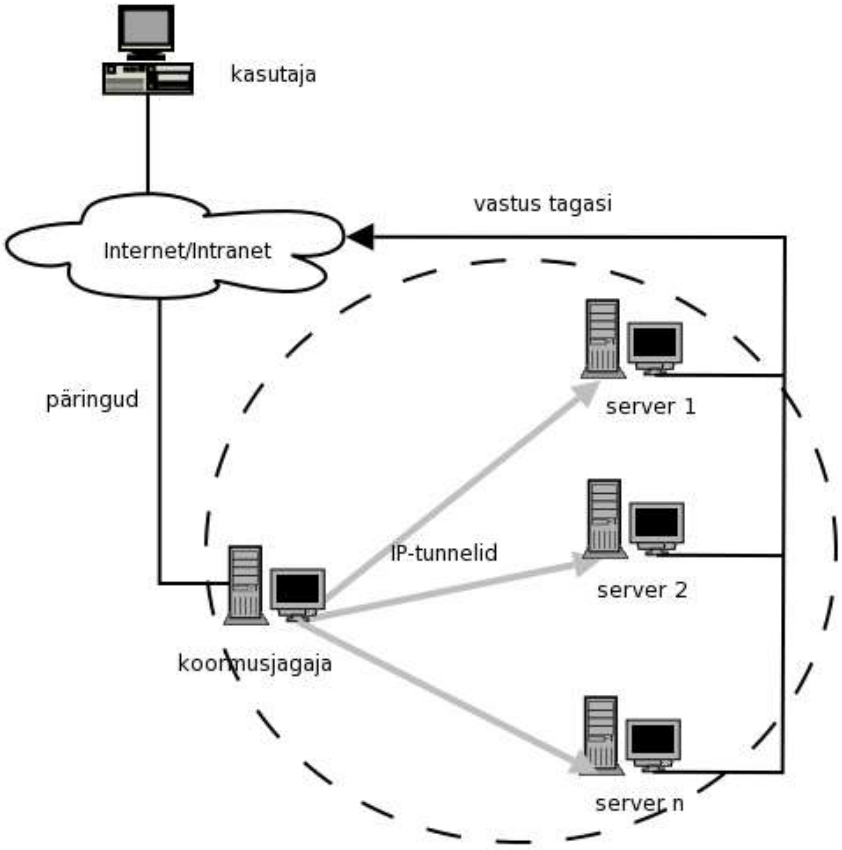
edasi sellele serverile. Kui reaalserver saab paketi, siis ta kapseldab selle lahti ja töötleb



Joonis 10: IP-tunneli vooskeem

päringu ning lõpuks saadab vastuse kasutajale vastavalt oma enda marsruutimistabelile. Kui ühendus lõpetatakse sihilikult või ajalimiidi ületamise (i.k. *timeout*) tõttu, siis kustutatakse vastav kirje ka ühendustetabelist. IP-tunneli vooskeem on kujutatud joonisel 10.

Tasub tähele panna, et reaalserverid võivad omada suvalist välist IP-aadressi suvalises alamvõrgus ning, et nad võivad geograafiliselt asuda suvalises punktis, kuid nad peavad toetama IP-kapseldamise protokolli. Nende tunneliseadmed peavad olema konfigureeritud nii, et süsteem suudaks vastu võetud paketti korralikult lahti kapseldada ning nad peavad omama ühte mitte-ARP (*Address Resolution Protocol*) võrguliidest, millele saaks konfigureerida virtuaalserveri IP-aadressi või peab süsteemi olema võimalik konfigureerida nii, et ta suunaks virtuaalserveri aadressile saadetud paketid kohalikku pidemesse (i.k. *socket*). Lähemalt teeme sellest probleemist juttu ühenduse protokollide peatüki lõpu poole.



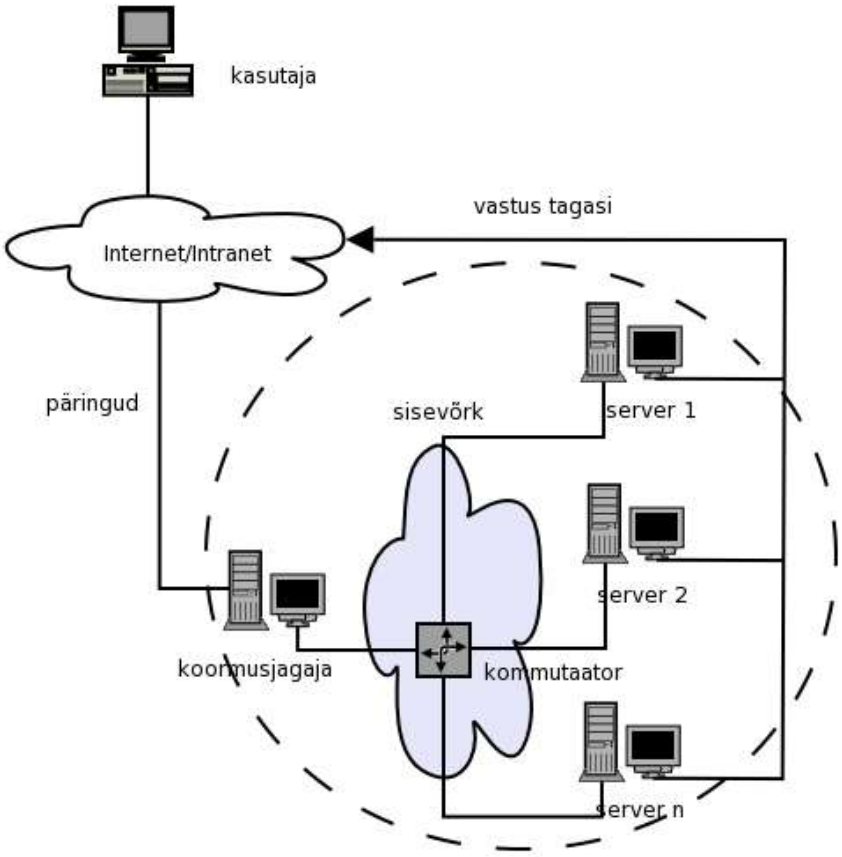
Joonis 11: IP-tunneli skeem

Kui kapseldatud pakett jõuab reaalserverisse, siis viimane pakib selle lahti ning leiab selle olevat saadetud virtuaalserveri aadressile, mis samas on ühtlasi ka tema ühe liidese aadress, seega töötleb ta paketti nii nagu oleks see tulnud otse talle ning saadab vastuse tagasi kasutajale otse, märkides lähteaadressiks virtuaalserveri IP-aadressi. IP-tunneldamist illustreerib joonis 11.

Hoolimata veidi keerukamast ülesehitusest võimaldab IP-tunnel tugevasti suurendada virtuaalserveri skaleeritavust.

2.7.2.5. Otsemarsruutimisega virtuaalserver

Selline päringu töötlemise tehnika vajab mingit dispetšerprogrammi, kuna virtuaalserveri aadressi jagatakse koormusjagaja ja reaalserverite vahel. Koormusjagaja omab võrguliidest, mis on konfigureeritud omama virtuaalserveri aadressi ning mida kasutatakse päringu pakettide vastuvõtuks. Koormusjagaja marsruudib paketid otse reaalserverile. Kõikidel reaalserveritel on üks mitte-ARP liides konfigureeritud omama virtuaalserveri aadressi või on nad konfigureeritud suunama virtuaalserveri aadressile saadetud paketid kohalikku pidemesse, seega suudavad nad töödelda pakette lokaalselt. Koormusjagaja ning



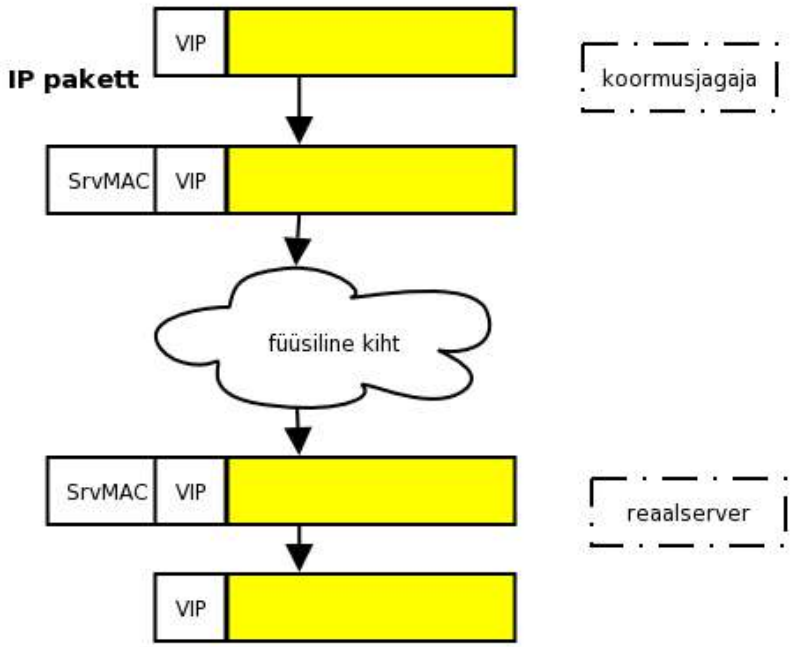
Joonis 12: Otsemarsruutimisega virtuaalserver

reaalserverite üks võrguliides peab olema füüsilises ühenduses jaoturi (i.k. *hub*) või kommutaatoriga (i.k. *switch*). Virtuaalserver otsemarsruutimise kaudu arhitektuur on kujutatud joonisel 12.

2.7.2.6.Kuidas toimib otsemarsruutimisega virtuaalserver

Siis, kui kasutaja pöördub virtuaalserveri poolt osutatava teenuse poole, saabub virtuaalserverile pakett. Koormusjagaja uurib paketi sihtaadressi ning väratit. Juhul, kui need sobivad virtuaalserveri poolt osutatava teenusega, valitakse vastavalt jagamise algoritmile välja mõni reaalserver ning ühenduse alustamise kohta lisatakse kirje ühenduste tabelisse. Seejärel saadab koormusjagaja ta otse valitud serverile. Kui mõni edaspidi sissetulev pakett kuulub selle ühenduse juurde ja valitud server on ühenduste tabelist leitav, suunatakse pakett otse edasi sellele serverile.

Kui reaalserver saab paketi, siis leiab selle olevat saadetud virtuaalserveri aadressile, mis samas on ühtlasi ka tema ühe liidese aadress, seega töötleb ta paketti nii nagu oleks see tulnud otse talle ning saadab vastuse tagasi kasutajale otse, märkides lähteaadressiks virtuaalserveri IP-aadressi. Kui ühendus lõpetatakse sihilikult või ajalimiidi ületamise tõttu, siis kustutatakse vastav kirje ka ühendustetabelist. Otsemarsruutimise vooskeem on kujutatud joonisel 13.



Joonis 13: Otsemarsruutimise vooskeem

Koormusjagaja lihtsalt muudab andmekaadri MAC aadressi valitud serveri omaks ning saadab selle siis lokaalvõrku. See on ka põhjuseks, miks koormusjagaja ning reaalserverid peavad ühte võrguliidest kaudu olema ühises lokaalvõrgu segmendis. Ka sellise ühendusmeetodi kõrvalnähtuseks võib olla ARP-probleem.

2.7.3.ARP probleem

Nii IP-tunneli kui ka otsemarsruutimise juures jagatakse virtuaalserveri aadressi koormusjagaja ja reaalserverite vahel. Mõnel juhul on võimalik olukord, kus reaalserverite IP-aadressid asuvad samas võrgusegmendis virtuaalserveri IP-aadressiga.

Juhul, kui nüüd reaalserver(id) vastavad virtuaalserveri asemel ARP-päringule, tekib olukord, kus virtuaalserverile mõeldud paketid saadetakse ühel hetkel koormusjagajale, järgmisel ühele reaalserverile ning siis võibolla hoopis teisele reaalserverile. Sellisel juhul tekib pakettidele vastamisel segadus, ühendus katkeb ning virtuaalserver lihtsalt ei funktsioneeriks. Seetõttu peab olema garanteeritud, et nii IP-tunneli kui ka otsemarsruutimise juures vastab ARP-päringutele ainult koormusjagaja, selleks, et saaks neid pakette vastu võtta ning edasi suunata virtuaalserveritele töötlemiseks. Samas ei tohi reaalserverid ARP-päringutele vastata, kuid nad peavad lokaalselt oskama töödelda virtuaalserveri IP-aadressile saabunud päringuid.

2.7.4.Protokollide võrdlus

Vastavalt eelpool esitatud virtuaalserveris kasutatavate ühendusprotokollide kirjeldustele võib koostada eeliste ja puuduste lühikirjelduse, mis on esitatud alljärgnevalt:

2.7.4.1.Network address translation

Eelised:

- reaalserverid võivad kasutada suvalist operatsioonisüsteemi, mis toetab TCP/IP ühendust;
- vajalik on ainult üks väline IP-aadress, reaalserverid toimivad sisevõrgus;

Puudused:

- reaalserverite hulk on piiratud, kuna nii päring kui ka vastus kirjutatakse koormusjagaja poolt ümber ning kui serverite arv ja/või edastatav andmemaht läheb väga suureks, muutub koormusjagaja enda koormus ülemäära suureks.

2.7.4.2.IP-tunneldamine

Eelised:

- reaalserverid saadavad päringute vastuseid kliendile otse, seega on võimalik päringute ja nende vastuste paketid juhtida mööda erinevaid marsruute;
- reaalserverid võivad asuda erinevates võrkudes (võrgusegmentides);

- suurendab märgatavalt virtuaalserveri skaleeritavust;

Puudused:

- reaalserverite operatsioonisüsteem peab toetama IP-tunneldamist.

2.7.4.3.Otsemarsruutimine

Eelised:

- reaalserverid saadavad päringute vastuseid kliendile otse, seega on võimalik päringute ja nende vastuste paketid juhtida mööda erinevaid marsruute;
- puudub tunneldamisel tekkiv pakettide ümbertöötlemisest tulenev koormus;
- puudub vajadus kasutada IP-tunneldamist toetavat operatsioonisüsteemi;

Puudused:

- reaalserverid peavad omama mitte-ARP võrguühendust või serverid tuleb konfigureerida nii et nad suunaks osa pakette kohalikku väratisse;
- nii serverid kui ka koormusjagaja peavad omama ühises võrgus vähemalt ühte võrguliidest;

	VS-NAT	**VS-tunnel**	**VS-Otsemarsruutimine**
Serveri operatsioonisüsteem	Suvaline	Tunneldav	Mitte-ARP seade
Serverite võrgu tüüp	Privaatne	LAN/WAN	LAN
Serverite hulk	Väike (10-20)	Suur (>100)	Suur (>100)
Serverite lüüs	Koormusjagaja	Oma marsruuter	Oma marsruuter

Tabel 2: Protokollide eelised ja puudused

Protokollide eelised ja puudused on esitatud kokkuvõtvalt tabelis 2. Nagu eelpool toodust nähtub, on omad eelised ja puudused igal ühendusprotokollil ning selle valik sõltub konkreetsest olukorrast.

2.8.Koormuse jagamise algoritmid

Kuna kasutaja jaoks on virtuaalserver läbipaistev, siis näeb ta ainult ühte masinat. Samuti ei oskaks ta üldjuhul ühenduda sama teenust pakkuva, kuid vähemkoormatud masina külge. Seega peab virtuaalserveris kasutaja päringute suunamine toimuma automaatselt, kusjuures on soovitav, et see suunamine arvestaks ka reaalserverite jõudlusega. Järgnevalt käsitlemegi virtuaalserverites kasutatavaid koormuse jagamise plaanurite algoritme.

2.8.1.Round-Robin algoritm

Round-Robin meetodi algoritm saadab iga järgneva siseneva päringu järgmisele serverile tema tabelis. Nii näiteks kolme reaalserveriga virtuaalserveris (serverid A, B ja C) läheb esimene päring serverile A, teine päring läheb serverile B, kolmas päring serverile C ning neljas päring läheb jälle serverile A, millega serverite "ringmäng" algab uuesti otsast peale. Selline meetod käsitleb kõiki reaalservereid võrdsetena, arvestamata sellega, kui palju on sissetulevaid päringuid, või kui palju ühe päringu töötlemiseks aega kulub. Praktikas näeb see välja nii, et DNS-nimeserver lahendab ühe domeeninime mitmeks erinevaks IP-aadressiks. Tänu algoritmi lihtsusele on virtuaalserveri juhtimisele kuluv ressurss väike, kuid samas on väga suur oht, et serverite omavaheline dünaamiline koormus läheb tasakaalust välja.

2.8.2.Kaalutud Round-Robin algoritm

Kaalutud Round-Robin algoritm on konstrueeritud selleks, et paremini toime tulla erinevaid jõudlusi omavate reaalserveritega. Igale reaalserverile on omistatud kaal – numbriline koefitsient, mis näitab tema jõudlust. Kaalu vaikeväärtuseks on 1. Kui näiteks serverid A, B ja C omavad vastavalt kaalude väärtusi 4, 3 ja 2, siis päringute jagamise jada peaks olema AABABCABC. Praktikas genereeritakse sellise koormuse jagamise meetodi korral reaalserverite koormuse jagamise jada igakordselt virtuaalserveri konfigureerimisel, vastavalt reaalserverite jõudluse kaalu väärtustele. Päringud jagatakse masinate vahel vastavalt genereeritud koormuse jagamise jadale samamoodi nagu round-robin meetodi korral. Sisuliselt on Round-Robin meetod kaalutud Round-Robin meetodi selline erijuht, kus kõik serverid on võrdse kaaluga.

Kaalutud Round-Robin algoritm on tavalisest Round-Robin algoritmist parem, kui serverite jõudlused on erinevad. Kuid juhul, kui päringute jõudlusevajadus on väga ebaühtlane, võib see viia serverite vahelise dünaamilise koormusjaotuse tasakaalust välja,

kuna on olemas võimalus, et suurem osa keerukatest päringutest suunatakse ühele ja samale reaalserverile.

2.8.3.Vähima ühenduse algoritm

Vähima ühenduse algoritm suunab päringu reaalserverile, millel olemasolevate ühenduste arv on kõige väiksem. Selline skeem on dünaamiline, kuna ta peab dünaamiliselt arvet iga masina küljes olevatest ühendustest. Vähima ühenduse meetod on sobiv, kui kõik reaalserverid on enam-vähem ühesuguse jõudlusega ning päringute jõudlusevajadus kõigub väga suurtes piirides. Sellise meetodi korral suunatakse päring vähima aktiivsete ühenduste arvuga reaalserverile.

Esimesel silmapilgul tundub, et vähima algoritm meetod toimib hästi ka siis, kui reaalserverite jõudlused on erinevad, kuna võimsamale arvutile suunatakse rohkem päringuid. Tegelikkuses ei toimu see päris nii TCP ühenduse TIME_WAIT oleku tõttu. TCP TIME_WAIT olek kestab tavaliselt 2 minutit, kuid selle kahe minuti jooksul saab populaarne veebiserver tuhandeid päringuid. Oletame, et server A on kaks korda võimsam, kui server B. Server A teenindab tuhandeid ühendusi ja hoiab neid TCP TIME_WAIT olekus, samas kui server B rabeleb oma tuhandete ühenduste lõpetamiseks. Seega ei sobi vähima ühenduse meetod virtuaalserverisse, kus on palju erinevate võimsustega reaalservereid.

2.8.4.Kaalutud vähima ühenduse algoritm

Kaalutud vähima ühenduse algoritm on vähima ühenduse meetodi versioon, kus igale serverile on võimalik omistada tema jõudluse kaal. Suurema jõudlusekaaluga reaalserverid võtvad suvalisel ajahetkel tehtud päringutest vastu suurema osahulga. Virtuaalserveri administraator saab igale reaalserverile omistada kaalu ning päringuid edastatakse igale masinale vastavalt tema ühenduste arvu ja tema võimsuse suhtele.

Võrreldes vähima ühenduse meetodiga vajab kaalutud vähima ühenduse meetod veidi rohkem serveriressurssi, kuna ta teostab jagamist. Kui reaalserverid on sarnase jõudlusega, siis, lootuses vähendada virtuaalserveri juhtimisele kuluvat ressurssi, on kaalutud vähima ühenduse algoritmi asemel kasutusel vähima ühenduse algoritm.

2.8.5.Oludele vastav vähima ühenduse algoritm

Oludele vastav vähima ühenduse algoritm on rohkem mõeldud sihtkoha IP koormuse ühtlustamiseks ning seda kasutatakse tavaliselt *cache* serverites. Selle meetodi algoritm

suunab mingi IP peale tehtud päringu seda IP-d teenindavale serverile ainult siis, kui see server on töökorras ja pole eriti koormatud. Juhul kui server on ülekoormatud (aktiivsete ühenduste arv on suurem, kui tema jõudluse koefitsient lubaks) ning süsteemis on vähemkoormatud server, siis suunatakse sellele IP-le tulevad päringud vähemkoormatud serverile.

2.8.6. Oludele vastav vähima ühenduse replikatsioonidega arvestav algoritm

Oludele vastav vähima ühenduse replikatsioonidega arvestav algoritm on samuti mõeldud sihtkoha IP koormuse ühtlustamiseks ning seda kasutatakse samuti tavaliselt *cache* serverites. Eelnevast meetodist erineb ta järgmiselt:

Koormusjagaja kaardistab mingi komplekti servereid, mis suudavad mingit teatavat ülesannet täita. Päring selle ülesande täitmiseks suunatakse vastavas serverite komplektis olevale arvutile vähima ühenduse algoritmi järgi. Juhul, kui kõik komplekti kuuluvad masinad on ülekoormatud, valib koormuse jagaja kogu virtuaalserveris vähim ühendusi omava serveri ning lisab selle ülesannet täitvate serverite komplekti. Kui serverite komplekti nimekirja pole mingi teatava aja jooksul muudetud, siis eemaldatakse kõige rohkem koormatud masin nimekirjast. See on vajalik selleks, et vältida võimalikku ühenduse dubleerimise ohtu.

2.8.7. Sihtaadressi räsitabeli algoritm

Sihtaadressi (i.k. *destination*) räsitabeli (i.k. *hash table*) meetodi algoritm suunab päringuid serveritele kasutades päringute sihtaadresside serverite IP-de järgi tehtud statistilist tabelit.

2.8.8. Lähteaadressi räsitabeli algoritm

Lähteaadressi (i.k. *source*) räsitabeli meetodi algoritm suunab päringuid serveritele kasutades päringute lähtekohta arvutite IP-aadresside järgi tehtud statistilist tabelit.

3. Praktiline osa

Eelnevas osas püüti anda lühiülevaate ajaloost ja tänapäevast aga samuti ka kasutatavast tehnoloogiast ja tarkvarast. Käesolevas osas kirjeldan oma kogemusi Zope klastri ülesseadmisel ja konfigureerimisel.

Praktilise osa eesmärgiks oli eelkõige klaster kokku seada, kasutades teoreetiliselt kogutud teadmisi aga samuti sellel klastril rakendustarkvarana seada toimima Zope veebiserver. Ühtlasi oli eesmärgiks uurida kas ja kui, siis kuipalju klastris päringule vastuse saamise kiirus (koguserveri jõudlus) suureneb.

3.1. Planeerimine

Klastri planeerimist alustasin tarkvara valikust. Kuna rakendustarkvarana oli kavas kasutada eelkõige Zope veebiserverit, siis tuli muude valikute osas lähtuda sellest, mida viimane võimaldab või toetab. Samuti oli soov uurida plaanurprogrammi toimimise praktilist poolt.

Zope veebiserver on avatud lähtekoodiga multiplatvormne tarkvara, seega on tema all toimiva operatsioonisüsteemi valik suhteliselt vaba. Samuti toetab Zope veebiserver klastril toimimist. Samas ei paku Zope vahendeid klastris koormuse jagamiseks erinevate aplikatsiooniserverite vahel.

Samuti oli soov, et kogu kasutatav tarkvara oleks vähemalt tasuta või veel parem kui see oleks ka avatud lähtekoodiga. Sellest nõudest tulenevalt jäid sõelale Linuxi ja BSD perekonda kuuluvad operatsioonisüsteemid. Lisaks eelpool toodud nõuetele tuli arvestada ka sellega, millist plaanurprogrammi kasutama hakata. Kuna Linuxit toetavaid programme oli rohkem, kui BSD-d toetavaid, siis jäi operatsioonisüsteemi valikuks Linux. Konkreetse distributsiooni valikul sai määravaks see, et Pedagoogikaülikooli platvormiks saab tulevikus Red Hat Enterprise Linux. Sellest tulenevalt sai operatsioonisüsteemiks valitud RHEL poolt vabaks arenduseks mõeldud versioon Fedora Core.

Konkreetse Zope serveri versiooni valikul tuli lähtuda tema sees toimivast rakendusest IVA. IVA viimane stabiilne versioon on numbriga 0.4.2. Vastavalt tema installeerimisdokumentatsioonile on selle versiooni puhul soovitav kasutada Zope versiooni 2.7.0, mis omakorda nõuab programmeerimiskeele Python versiooni 2.3.3

Plaanurprogrammi valikul nii rangeid nõudeid ei olnud. Seega otsustasin kasutada Linuxi

kernelisse kompileeritavat lisamoodulit LinuxVirtualServer ning võimalust katsetada ka mõnda teist klastri moodustamiseks ja haldamiseks mõeldud tarkvarakomplekti.

Lõppkokkuvõttes jäi sõelale järgmine tarkvarakomplekt:

- operatsioonisüsteem – Linux Fedora Core 1;
- rakendustarkvara – Zope 2.7.0 koos IVA 0.4.2 mooduliga;
- plaanurprogramm – LinuxVirtualServer koos reservatsiooniga võimalusel testida teisi variante;

Riistvara poolel eriliselt suurt valikut ei olnud. Kasutada õnnestus Pedagoogikaülikooli riistvaralaboris olevat vanemat arvutustehnikat. Mingil määral oli selline piirang isegi hea, sest lahendus, kus on kasutada oma omadustelt kehvemat riistvara, peaks koormuste erinevused rakendustarkvara kasutamisel rohkem välja tooma. Samuti võib minu arvates sellise riistvara koosluse toimimisest teha järeldusi, milline peaks olema riistvara tegelikus tööolukorras suurte koormuste juures.

3.2.Riistvara

Riistvaralaboris olevatest komponentidest õnnestus lõppkokkuvõttes kokku seada neli masinat. Arvutite konfiguratsioon on toodud tabelis 3.

	masin1	***masin2***	***masin3***	***masin4***
Emaplaat	intel ca810c	jetway 615tcs	Intel 845	Tomato
Protsessor	P3 500e	celeron 800	celeron 1800	p166MMX
Kõvaketas	wd644 (6,4 GB)	wd644 (6,4 GB)	(20 GB)	wd644 (6,4 GB) + quantum fireball (3,2 GB)
Operatiivmälu	64+32 PC100	128 PC100	512MB	128 PC100
Võrgukaart	Rtl8029	rtl8139c	Intel, 100Mb/s	rtl8139c
Lisa	CD-ROM	CD-ROM		CD-ROM

Tabel 3: Kasutusel olnud riistvara

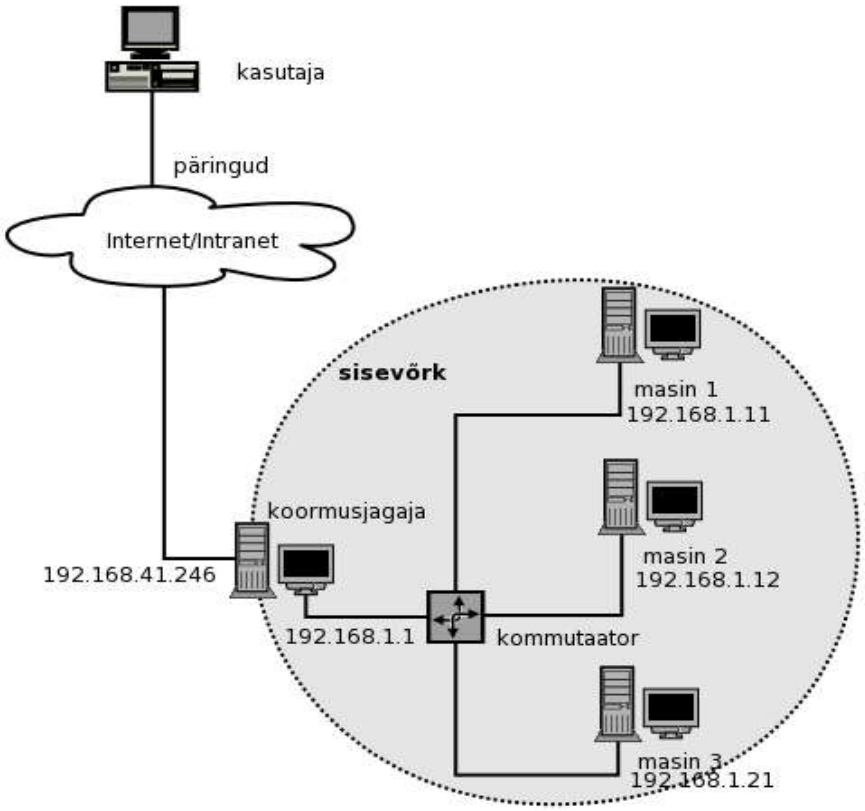
Nagu tabelist näha, on tegemist väga erineva riistvaraga, samas on see testserveri juures hea, kuna näitab, kuidas Linuxi virtuaalserver tegelikkuses erineva riistvaraga toime tuleb. Samuti tuli sellega arvestada virtuaalserveri planeerimisel ning arvutitele ülesannete määramisel. Lähtuvalt arvutite eeldatavast jõudlusest, planeerisin ülesanded nii, et masinad 1 ja 2, kui jõudluselt enam-vähem võrdsed, täidavad aplikatsiooniserveri ülesannet, masin 3 on aplikatsiooniserver ja ühtlasi ka andmebaasi server ning masin 4 täidab marsruuteri ja koormusjagaja ülesannet.

3.3. Võrk

Arvutite omavahelise ühenduse meetodiks sai valitud NAT ja seda järgnevatel põhjusel:

- lokaalne võrk on kiirusega 10 MB/s, kuid meil on vaja reaalserverite vahele kiiret ühendust;
- masinate hulk ei ole suur, samuti ei saa ta suur olema reaalsetes tingimustes – tuletame meelde, et NAT puhul peaks masinate arv jääma alla 20-ne;
- kõik reaalserverid asuvad füüsiliselt ühes kohas.

Füüsiliselt sai võrk üles ehitatud nii, nagu on kujutatud joonisel 14.



Joonis 14: Virtuaalserveri topoloogiline skeem

Klastri planeerimisel oli võimalus kasutada nelja Intraneti aadressi -192.168.41.246-249. Edaspidi kasutan TPÜ sisevõrgu suhtes mõisteid intranet või välisvõrk ning virtuaalserveri sees olevate mõistet sisevõrk. Samas sisevõrgu plaanimisel olid mul suhteliselt vabad käed. Kuna reaalservereid ei ole palju, siis sai valitud C-klassi aadressruum võrguaadressiga 192.168.1.0 ning serverid said järgneva konfiguratsiooni.

```
###  KLASTRI KONFIGURATSIOON  ###
Väline IP   = 192.168.41.246 (247, 248, 249)
DNS         = 193.40.239.1
GateWay     = 192.168.41.1
NetMask     = 255.255.255.0

Sisevõrk
nat+ruuter+switch       cluster.cluster.sise    192.168.1.1
aplikatsiooniserver 1   masin1.cluster.sise     192.168.1.5
aplikatsiooniserver 2   masin2.cluster.sise     192.168.1.6
andmebaasi server       db.cluster.sise         192.168.1.10

sisevõrguaadress  = 192.168.1.0
DNS                     = 192.168.1.1
GateWay           = 192.168.1.1
NetMask           = 255.255.255.0
```

3.4. Operatsioonisüsteemi installeerimine ja konfigureerimine

Nagu juba eelpool mainitud, sai arvutite operatsioonisüsteemiks valitud Fedora Core 1 ning kuigi NAT ühenduse puhul ei ole serverite operatsioonisüsteem oluline, sai kõik masinad installeeritud ühe ja sama operatsioonisüsteemiga. Installeerimispakettide valikul lähtusin sellest, et tegemist on võrguarvutiga ning et tegemist on arendatava süsteemiga. Seega sai valitud kõik selleks vajalik.

Peale operatsioonisüsteemi installeerimist püüdsin käivitada Linux Virtual Serverit, kuid selgus, et see ei ole Fedora Core vaikekoosseisu lülitatud. Samuti selgus, et virtuaalserveri ehitamiseks vajalikke komponente ei ole ka Fedora Core kerneli lähtekoodis. Seega osutus vajalikuks alla tõmmata ja installeerida uus kernel.

Uueks kerneliks sai valitud Linuxi standardkernel versiooninumbriga 2.4.26, mis sai allatõmmatud aadressilt http://www.kernel.org. Selles kernelis on põhiline virtuaalserveri ehitamiseks vajalik tarkvara juba sees, kuid vajalik on tähelepanna asjaolu, et kerneli vanemad versioonid võivad enne konfigureerimist ja kompileerimist vajada ka paikamist (*patch*). Vajalik paik leidub http://www.linuxvirtualserver.org tarkvarasektsioonist. Kerneli kompileerimine ja installeerimine toimub täpselt samamoodi, nagu tavaliselt, kuid tema konfigureerimisel tuleb tähelepanu pöörata virtuaalmasina seksioonile.

```
Code maturity level options --->
  [*] Prompt for development and/or incomplete code/drivers

Networking options --->
  [*] Network packet filtering (replaces ipchains)
  [ ]   Network packet filtering debugging
  ...
    IP: Netfilter Configuration  --->
    IP: Virtual Server Configuration  --->
      <M> virtual server support (EXPERIMENTAL)
      [*]  IP virtual server debugging
      (12) IPVS connection table size (the Nth power of 2)
      --- IPVS scheduler
      <M>  round-robin scheduling
      <M>  weighted round-robin scheduling
      <M>  least-connection scheduling scheduling
      <M>  weighted least-connection scheduling
      <M>  locality-based least-connection scheduling
      <M>  locality-based least-connection with replication scheduling
      <M>  destination hashing scheduling
      <M>  source hashing scheduling
      --- IPVS application helper
      <M>  FTP protocol helper
```

Kerneli konfigureerimise utiliidi saab välja kutsuda käsuga

```
[cluster]$# make xconfig
```

Virtuaalserveri marsruutimistabelite muutmiseks on vajalik eraldi alla tõmmata ja kompileerida moodul nimega ipvsadm. Antud juhul sai see tõmmatud veebilehelt http://www.linuxvirtualserver.org/software/ipvs.html#kernel-2.4 Mooduli kohta saab vajalikku lisainformatsiooni temaga kaasatulevast dokumentatsioonist käsuga

```
[cluster]$# man ipvsadm
```

3.5.Plaanuri installeerimine

Järgnevalt vaatame programmi Keepalived installeerimist ja konfigureerimist. Keepalived on mõeldud lihtsaks ja töökindlaks plaanur- ning jälgimisprogrammiks Linuxi Virtuaalserveris. Programm on kirjutatud programmeerimiskeeles C. Keepalived võimaldab:

- seada arvuti väliseid IP-aadresse süsteemsetest seadetest sõltumatult ning ilma süsteemseid seadistusvahendeid kasutamata;
- muuta virtuaalserveri marsruutimistabeleid;
- käsitleda koormusjagaja ja/või terve virtuaalserveri töö katkestusi;
- kontrollida reaalserverite olukorda ning vajadusel muuta koormusjagaja marsruutimistabelit vastavalt tegelikule olukorrale;
- saata virtuaalserveri administraatorile teateid reaalserverite töö katkemise kohta;

- muuta virtuaalserveri konfiguratsiooni ilma virtuaalserveri süsteemseid vahendeid kasutamata.

Konkreetse Keepalived versioon oli 1.1.7 ning see sai tõmmatud aadressilt http://www.keepalived.org/ Enne Keepalived konfigureerimist ja kompileerimist on vajalik konfigureerida ja kompileerida kernel. Keepalived enda konfigureerimine ja kompileerimine toimub järgnevalt:

```
[cluster]$# tar xzvf keepalived-1.0.1.tar.gz
cd keepalived-1.0.1
./configure
make
make install
```

Keepalived konfigureerimise ja installeerimise juures on vajalik tähele panna asjaolu, et ta vajab selleks kerneli lähtekoodi ning otsib seda kataloogist /usr/src/Linux/ Peale programmi kompileerimist võib osutuda vajalikuks üle vaadata programmi enda ja tema konfiguratsioonifaili asukoht. Programm peab asuma /sbin/keepalived ning ta otsib oma konfiguratsioonifaili kataloogist /etc/keepalived/ Programm tekitab lähtekoodi kataloogi konfigureerimise ja kompileerimise logi, mille järgi on mugav vigu otsida, juhul, kui midagi ei tööta või mõnda installeeritud komponenti ei leia.

3.6.Plaanuri konfigureerimine

Võrgu konfiguratsioonist oli eelnevalt juttu, kuid kordame selle veelkord üle:

```
###  KLASTRI KONFIGURATSIOON  ###
Väline IP   = 192.168.41.246 (247, 248, 249)
DNS         = 193.40.239.1
GateWay     = 192.168.41.1
NetMask     = 255.255.255.0

Sisevõrk
nat+ruuter+switch       cluster.cluster.sise    192.168.1.1
aplikatsiooniserver 1   masin1.cluster.sise     192.168.1.5
apikatsiooniserver 2    masin2.cluster.sise     192.168.1.6
andmebaasi server       db.cluster.sise         192.168.1.10

sisevõrguaadress  = 192.168.1.0
DNS                     = 192.168.1.1
GateWay           = 192.168.1.1
NetMask           = 255.255.255.0
```

Konfiguratsioonifailis loetakse kommentaariks kõike, mis algab märgiga # või ! ning seda kuni rea lõpuni. Järgnevalt toon konfiguratsioonfaili näite, koos omapoolsete kommentaaridega:

```
#    SEE ON KOMMENTAAR
! KA SEE ON KOMMENTAAR
```

Konfiguratsioonifaili esimene plokk on globaalsete definitsioonide oma. Siin määratakse ära, kuidas ja kuhu saadetakse veateated serveri tõrke korral.

```
global_defs {
  notification_email {
    meelis@tpu.ee    # saadame maili juhendajale
    hillarp@tpu.ee   # ja mulle ka
  }
   notification_email_from cluster@tpu.ee
   smtp_server 193.40.239.27  # SMTP server on lin2
   smtp_connect_timeout 30
   lvs_id test_cluster
}
```

Sektsioon *notification_email* on mailiaadressite jaoks, kuhu serveri tõrke korral veateade saadetakse, võtmesõna *smtp_server* näitab, millise SMTP serveri kaudu virtuaalserver oma veateate peab saatma ning võtmesõna *lvs_id* on virtuaalserveri identifikaator. Selliste seadetega virtuaalserveri poolt saadetud teade näeb välja nii:

```
Date: Thu, 29 Apr 2004 21:14:23 +0300
From: cluster@tpu.ee
To: hillarp@tpu.ee
Subject: [test_cluster] Realserver 192.168.1.12:8080 - DOWN

=> CHECK failed on service : recevice data <=
```

VRRP (*Virtual Router Redundancy Protocol*) sektsioon on mõeldud juhuks, kui virtuaalserveril on olemas ka tagavara marsruuter-koormusjagaja. Juhul, kui põhiline marsruuter, mingil annab põhjusel tõrke, siis võtab tagavara marsruuter tema ülesanded üle. See sektsioon on vajalik täita sõltumata sellest, kas tagavara marsruuter tegelikult eksisteerib või mitte.

```
vrrp_sync_group VG1 {
   group {
      VI_1
      VI_GATEWAY
   }
}
```

Järgmine sektsioon on *vrrp_instance*. Iga marsruuteril kasutatav võrguliides peab kuuluma vähemalt ühte sektsiooni. Samas võib neid sektsioone olla kuitahes palju. Ühes sektsioonis on koos võrguliidesed, mis loogiliselt kuuluvad kokku.

VRRP sektsioonis olevate võtmesõnade tähendus on järgnev:

Võtmesõna *state* määrab ära, kelleks marsruuter arvab end olema. Tegelikult määrab selle,

kes *masteriks* valitakse, võtmesõna *priority* olev number. Seega isegi juhul, kui võtmesõnas *state* on määratud olekuks *MASTER* kuid võtmesõna *priority* järel on väiksem number võidab valimised teine marsruuter. Kahe marsruuteri korral on soovitav *MASTER* ja *BACKUP* vahele jätta vähemalt 50 ühikuline vahe. Kui tihti valimised toimuvad määrab ära võtmesõna *advert_int* taga olev number (sekundites). Võtmesõna *smtp_alert* ütleb, et määratud aadressile saadetakse teade, kui konkreetne masin muudab oma *MASTER* - *BACKUP* olekut. Sektsioon *authentication* on põhi- ja varumarsruuteri(te) omavahelise sünkroniseerimise jaoks.

```
vrrp_instance VI_1 {
    state MASTER
    interface eth0
    smtp_alert
    virtual_router_id 51
    priority 100
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
```

Järgnev *virtual_ipaddress* plokk näitab millistel IP-aadressidel virtuaalserver vastab. Kui siin plokis on mõni vajalik IP-aadress määramata, siis seda aadressi ei kuulata. Tasub tähele panna, et selles plokis olev IP-aadress ei pea olema tegelik võrgukaardi aadress. Reaalserverid asuvad tegelikult siinses plokis toodud IP-aadressi taga.

```
        virtual_ipaddress {
                192.168.41.246
        ! siia võib neid vajadusel veel lisada
        }
}
```

Järgnevaks määrame ära reaalserverite vaikevärati, mis meie näite puhul oli 192.169.1.1.

```
vrrp_instance VI_GATEWAY {
        state MASTER
        interface eth1
        lvs_sync_daemon_inteface eth1
        virtual_router_id 52
        priority 150
        advert_int 1
        smtp_alert
        authentication {
                auth_type PASS
                auth_pass example
        }
        virtual_ipaddress {
                 192.169.1.1
        }
}
```

Järgnevas plokis määrame ära virtuaalserveri parameetrid. Selliseid plokke võib olla

kuitahes palju. Meie näites on vajalik suunata kogu 80-sse väratisse sisenev liiklus ümber reaalserverite väratisse 8080.

```
virtual_server 192.168.41.246 80 {
    delay_loop 6
    lb_algo rr
    lb_kind NAT
    persistence_timeout 50
    protocol TCP
```

Võtmesõna *delay_loop* ütleb, kui sageli reaalservereid kontrollitakse (sekundites), *lb_algo* ütleb millist algoritmi koormuse jagamise juures kasutatakse. Võimalikud väärtused on rr (*Round-Robin*), wrr (*Weighted Round-Robin*), lc (*Last Connection*) ja wlc (*Weighted Last Connection*). Võtmesõna *lb_kind* ütleb millist ühendust kasutatakse. Siin on võimalikeks väärtusteks NAT, DR ja TUN. Võtmesõna *persistence_timeout* määrab aja, mille järel ühelt ja samalt IP-aadressilt tulevad päringud suunatakse uuele reaalserverile ning võtmesõna *protocol* määrab, millist protokolli kasutatakse.

```
    sorry_server 192.168.41.246 90
```

Võtmesõna *sorry_server* ütleb, kuhu kliendi suunatakse päringud, kui tema poolt küsitud teenust pole võimalik osutada. Selleks võib olla ka koormusjagaja ise, eeldusel et temal töötab mingisugune veebiserver.

Järgnevas seksioonis määrame ära reaalserveri parameetrid.

```
  real_server 192.168.1.11 8080 {
    weight 1
      HTTP_GET {
            url {
              path /
              digest b3582fa4b2738526fe194ff39686c715
            }
         connect_timeout 3
         connect_port 8080
         nb_get_retry 3
         delay_before_retry 3
      }
  }
```

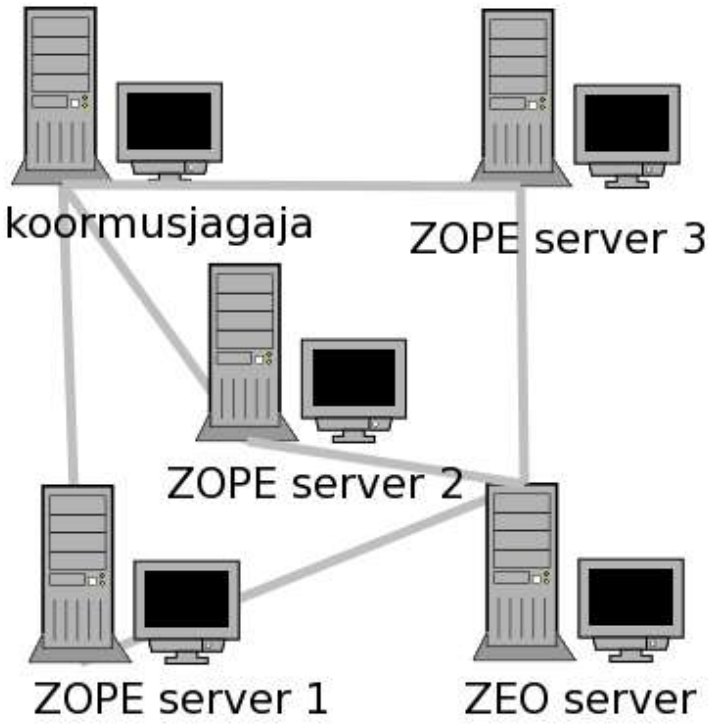
Võtmesõna *weight* ütleb, milline on serveri kaal valitud algoritmi juures. Sektsioon *HTTP_GET* ütleb, millist meetodit tuleb kasutada reaalserveri oleku kontrolliks. Võimalikud valikud on lisaks veel *SSH_GET* ja *TCP_CHECK*. Viimase kahe korral muutuvad ka järgnevad read. Võtmesõna *path* määrab ära teekonna testleheküljeni. Mina kasutasin konkreetsel juhul Zope serveri esilehe vaikeväärtust. Võtmesõna *digest* võimaldab krüptoalgoritmi MD5 kasutades kontrollida, kas leht mida näidatakse on õige,

kuid see võtmesõna toimib ainult plokis *HTTP_GET* ja *SSH_GET*. Konkreetselt siin käsitletud juhul seda võimalust ei kasutatud. Võtmesõnad *connect_timeout, connect_port, nb_get_retry, ja delay_before_retry* määravad ära parameetrid, mille järel saadetakse teade reaalserveri tõrke kohta. Selliseid plokke peab olema vastavalt reaalserverite arvule. Täpsemad juhendid aga samuti ka näidiskonfiguratsioonifailid on Keepalived[24] kodulehel ning programmiga kaasatulevas dokumentatsioonis. Konkreetselt selles klastris kasutatud Keepalived konfiguratsioonifail on käesoleva bakalaureusetöö lisa 1.

3.7.Zope veebiserveri ja IVA installeerimine

Zope veebiserveri installeerimist ei ole käesolevas töös peetud vajalikuks kirjeldada, kuna vastavasisuline põhjalik juhend versiooni 2.6 kohta on ära toodud ZopeBook-is[25] Küll aga räägitakse seoses mitme paralleelse aplikatsiooniserveri olemasoluga ja andmebaasiserveri viimisega erinevatesse masinatesse tekkivatest erisustest. Samuti sellest, kuidas aplikatsiooniserver andmebaasiserveriga koos tööle saada.

Zope andmebaasiserverit nimetatakse ZEO (*Zope Enterprise Objects*) ning selles hoitakse veebiserveri toimimiseks vajalikke andmeid. Zope klastri loogiline skeem on kujutatud joonisel 15.



Joonis 15: ZEO serveri ja ZEO klientide põhimõtteskeem

Kõigepealt märgin ära, et versiooni 2.7 puhul on ZEO installeerimine ZopeBook-is

esitatust veidi erinev. Nimelt pole ZEO-d enam eraldi alla laadida, sest ta on olemas Zope installatsioonipakis. ZEO installeerimiseks pole vaja teha muud, kui peale Zope installeerimist minna kataloogi $ZOPEHOME/bin ning anda käsk

```
./mkzeoinstance --prefix=/zeo/instance/asukoht
```

Soovitavalt peaks see kataloog asuma kusagil /var/ kataloogi all. ZEO konfigureerimiseks tuleb muuta /zeo/instance/asukoht/etc all olevat zeo.conf nimelist faili. Järgnevalt on toodud näiteks käesoleva bakalaureusetöö raames kasutatud konfiguratsioonifail, koos minupoolsete kommentaaridega.

```
# ZEO configuration file

%define INSTANCE /var/zope/ZEOinstance

<zeo>
  address 7700
  read-only false
  invalidation-queue-size 100
  # monitor-address PORT
  # transaction-timeout SECONDS
</zeo>
```

Siinses plokis tähistab võtmesõna *address* väratit (*port*) millel ZEO andmebaasiserver kuulama hakkab. See võib olla masina suvaline vaba värat. Võtmesõna *read-only* määrab ZEO andmebaasi ainult loetavaks. Võtmesõna *invalidation-queue-size* määrab maksimaalse järjekorras olevate päringute arvu. Juhul, kui serveris on hulgaliselt väikeseid faile, mille poole sagedasti pöördutakse, on kasulik see number suuremaks muuta.

```
<filestorage 1>
  path $INSTANCE/var/Data.fs
</filestorage>
```

Selles seksioonis määrame ära, kus asub ZEO andmebaasifail ning järgnevas selle, kus asub ZEO logifail.

```
<eventlog>
  level info
  <logfile>
    path $INSTANCE/log/zeo.log
  </logfile>
</eventlog>
```

Plokis *runner* määrame ära vajalikud parameetrid ZEO käivitamiseks deemonina (*daemon*). Siinjuures on vajalik tähele panna, et logifaili asukoht oleks õige. See peab olema sama mis on määratud eelnevas lõigus, kuna ZEO-d käivitav skript ei oska logifaili puudumisel seda ise tekitada.

```
<runner>
  program $INSTANCE/bin/runzeo
  socket-name $INSTANCE/etc/zeo.zdsock
  daemon true
  forever false
  backoff-limit 10
  exit-codes 0, 2
  directory $INSTANCE
  default-to-interactive true
  # user zope
  python /usr/bin/python2.3
  zdrun /var/zope/lib/python/zdaemon/zdrun.py

  logfile $INSTANCE/log/zeo.log
</runner>
```

Sellega on ZEO server konfigureeritud. ZEO serveri käivitamiseks tuleb minna kataloogi / zeo/instance/asukoht ja anda käsk

```
./zeoctl start
```

Zope serveri ja ZEO serveri koostööks on vajalik muuta Zope serveri konfiguratsioonifaili, mis asub kataloogis /zeo/instance/asukoht/etc Täpsemalt on vaja muuta konfiguratsioonifaili lõpus asuvat „Database (zodb_db) section“ all olevat osa. Seal tuleb välja asendada lõik

```
<zodb_db main>
    # Main FileStorage database
    <filestorage>
      path $INSTANCE/var/Data.fs
    </filestorage>
    mount-point /
</zodb_db>
```

alljärgnevaga

```
 <zodb_db main>
   mount-point /
   <zeoclient>
     server 192.168.1.21:7700
     storage 1
     name zeostorage
     var $INSTANCE/var
   </zeoclient>
 </zodb_db>
```

Tähelepanu tuleb pöörata võtmesõna *server* taga olevale väärtusele. Seal on antud IP aadress ja kooloni taga värat, millel Zope server oma andmebaasiga ühendust võtab. Aadress peab olema ZEO serveri oma ning värati number peab olema sama, mis on määratud ZEO serveri konfiguratsioonifailis. Testimiste juures kasutatud konfiguratsioonifail terviklikul kujul on käesoleva töö lisa 2.

IVA installeerimine toimub täpselt nii, nagu on kirjutatud IVA installeerimisjuhendis. Ainuke erinevus, seoses mitme aplikatsiooniserveriga, on see, et IVA peab olema installeeritud kõikidesse Zope serveriga masinatesse, kusjuures tuleb tähelepanu osutada sellele, et kõigis serverites oleks üks ja sama IVA versioon. ZEO serverile IVA installeerimine ei ole vajalik.

3.8. Testimine

Testimiseks kasutasin eraldi arvutit, millel oli kaks võrgukaarti. Võrgukaardid konfigureerisin nii, et üks neist sai aadressi 192.168.41.248 ja teine 192.168.41.249. Testimise ajaks tekitasin IVA-sse kaks kasutajat nimedega tester1 ja tester2, samuti sai samanimelised kasutajad loodud testarvutisse. Lisaks kasutajatele sai IVA-sse loodud üks testkursus, kuhu mõlemad kasutajad ka osalejaks registreeriti. Lisaks sai mõlema kasutaja töölauale laetud TestFail_1, suurusega 14,96 MB. Samuti sai Zope juurkataloogi tekitatud kaust, kuhu sai laetud TestFail_2, suurusega 69 MB. Testimise juures kasutasin brauseritena Mozillat, Konquerori ja Lynxi. Koormustestiks kasutasin wget nimelist utiliiti koos järgnevate võtmetega.

```
wget --output-file=logfile.log
     --bind-adress=192.168.41.248
     --server-response
     --http-user=kasutaja
     --http-passwd=salasõna
     --cache=off
     --tries=10000
     --recursive
     http://192.148.41.246/IVA
```

Programmil on võtmesõnad *verbose* (maksimaalne väljund) ja *timestamp* (ajatempel) vaikimisi alati tõesed, seega polnud vajadust neid eraldi ära määrata. Võtmesõna *output-file* määrab, millisesse faili logitakse tegevused. Võtmesõna *bind-adress* määrab, millist aadressi tegevuse juures kasutatakse. Kasutaja tester1 korral oli aadressiks 192.168.41.248 ning tester2 korral 192.168.41.249. Võtmesõnad *http-user* ja *http-passwd* määravad ära IVA-sse sisselogimiseks vajalikud kasutajanimed ja salasõnad. Võtmesõna cache määrab, kas programm kasutab oma tegevuse juures vahemäluserverit. Testide juures lülitasin ta valikuga *off* alati välja. Võtmesõna *tries* määrab katsete arvu, kui mitu korda püütakse mingit veebilehte tema mittesaamisel uuesti laadida. Vaikimisi on siin väärtuseks 1 ning väärtus 0 määraks, et lehte püütakse laadida lõpmatult. Võtmesõnaga *recursive* määrasin, et alla tõmmataks rekursiivselt kõik veebilehed ning kõige lõpuks on aadress, kustkohast wget oma tegevust alustab.

Seda, kuidas toimib koormusjagaja ning millisele reaalservereile ta mingi välise ühenduse suunas vaatasin käsuga

```
[cluster]$# ipvsadm -L -n -c
```

Kusjuures võti L tähendab siin nimekirja (*list*), võti n tähendab „mitte lahendada numbrilisi aadress nimedeks“ ning võti c „näidata kehtivaid ühendusi“.

Koormusjagajat konfigureerisin enne igat testi vastavalt vajadusele. Selleks muutsin Keepalived konfiguratsioonifaili ning peale muudatuste tegemist andsin käsu

```
[cluster]$# service keepalived restart
```

Testides osalevate masinate hetkekoormust jälgisin utiliidiga top. Kõikides testides oli wget poolt allatõmmatavate failide kogumahuks 110 MB.

3.9.Test 1

Selles testis proovisin, kuidas toimib kasutajaliides ning kuidas koormusjagaja. Klaster oli seadistatud nii, et kõik reaalserverid olid võrdsete õigustega ning et toimisid kõik kolm aplikatsiooniserverit ja andmebaasiserver.

Logisin IVA-sse sisse kahe erineva kasutajana ning ühtlasi ka administraatorina. Koormusjagaja suunas kõik kolm ühendust erinevatesse serveritesse. Administraatorina moodustasin uue kursuse, ning lisasin ühe kasutaja koheselt selle osalejate nimekirja. Teise kasutajana registreerisin ennast sellele kursusele ja kui administraator oli kasutaja vastuvõtnud, siis lugesin postkastist sellekohast teadet. Mõlema kasutaja töölauale lisasin eelpool nimetatud faili. Administraatorina tekitasin Zope serveri juurkataloogi testkataloogi kuhu lisasin eelpool mainitud TestFail_2 nimelise faili.

Kogu kasutajate ja administraatori tegevus toimis normaalselt ja ilma erilise viivitusteta. Aplikatsiooniserverite protsessorikoormus püsis 70% lähistel ning protsessori keskmine koormus (i.k. *load*) oli alla ühe. Andmebaasiserveri koormus oli vahemikus 2-3%. Aplikatsiooniserverite koormus tõusis umbes 90%-ni siis, kui toimus failide üleslaadimine. Andmebaasi serveri koormus tõusis ligikaudu 5%-ni.

3.10.Test 2

Selle testi jaoks seadistasin klastri nii, et kõik päringud suunataks masinale 3. Selline seadistus on võrdne olukorraga, kus kasutusel on ainult üks Zope server ilma ZEO-ta. Testis käivitasin ühe wget-i eelpool toodud seadistustega.

Test kestis 34 minutit ning sellejuures koormas Zope masinat 70-80% ning ZEO 1-2%,

kusjuures mälu vajasid nad vastavalt 80-90% ja 1-2%. Masina üldine koormus kõikus vahemikus 0,92-1,17.

3.11.Test 3

Selle testi juures oli klaster seadistatud samuti nagu eelmise testi puhul, kuid testmasinas käivitasin kummagi kasutaja alt eraldi wget-i.

Sellest testis kestis tester1 alt käivitatud wget 84 minutit ning tester2 alt käivitatud wget 86 minutit. Seejuures tarbis Zope 95% protsessori võimsusest ning ZEO umbes 3% protsessori võimsusest. Mälu vajasid mõlemad samapalju, kui eelmises testis. Masina üldine koormus oli 1,97-2,19.

3.12.Testid 4 ja 5

Nende testide ülesandeks oli võrrelda aplikatsiooniserverite jõudlust omavahel. Klaster oli testide juures seadistatud vastavalt nii, et neljandas testis suunati päringud masin1 peale ning viiendas testis masin2 peale. Andmebaasiserverina toimis mõlemail juhul masin3. Kummaski testis käivitasin ainult ühe kasutaja alt ühe wget-i.

Mõlemais testis olid tulemused sarnased. Esimesel juhul kestis test 38 minutit, teisel juhul 37 minutit, kusjuures Zope vajas mõlemal juhul protsessorivõimsust 70-80% ning mälu 80-90% masina üldine koormus oli mõlemal juhul 0,8 ja 0,9 vahel. ZEO vajas mõlemas testis protsessorivõimsust ligikaudu 1% ning mälu 1-2%.

3.13.Testid 6 ja 7

Nende testide ülesandeks oli võrrelda masinate 1, 2 ja 3 jõudlust omavahel. Klaster oli testide juures seadistatud vastavalt nii, et kuuendas testis suunati päringud masin1 peale ning seitsmendas testis masin2 peale. Andmebaasiserverina toimis kummaski testis vastavasse masinasse seadistatud ZEO server. Kummaski testis käivitasin ainult ühe kasutaja alt ühe wget-i.

Samuti nagu eelmises testis, olid ka siin mõlemais testis tulemused sarnased. Esimesel juhul kestis test 76 minutit, teisel juhul 74 minutit, kusjuures Zope vajas mõlemal juhul protsessorivõimsust 70-80% ning mälu 80-90% masina üldine koormus oli mõlemal juhul alla 0,9 ja 1,1 vahel. ZEO vajas mõlemas testis protsessorivõimsust ligikaudu 1% ning mälu 1-2%.

3.14.Test 8

Selles testis oli klaster seadistatud nii, et kõik päringud jagataks võrdselt masinale 1 ja masinale 2 ning masin 3 täidaks ainult andmebaasiserveri ülesannet. Testarvutis käivitasin kummagi kasutaja alt ühe wget-i protsessi, kusjuures seadistasin wget-i nii, et tester1 ütleks oma IP-aadressiks 192.168.41.248 ja tester2 192.168.41.249. Koormusjagaja suunas tester1 masina 1 külge ning tester2 masina 2 külge.
Tester1 alt käivitatud wget kestis 67 minutit ning tester2 alt käivitatud wget 54 minutit. Seejuures oli mõlemal aplikatsiooniserveril enamvähem võrdne koormus. Mõlema Zope vajas protsessorijõudlust 70-80% ning mälu 80-90%, mõlema masina üldine koormus kõikus 0,7 ja 0,9 vahel. ZEO server kasutas selle testi juures protsessori võimsust 5-6% ning mälu 1-2%.

3.15.Järeldused testidest

Testidest korrektsete järelduste tegemiseks oleks vajalik vaadata Riistvara peatükis toodud tabelit 3.

Riistvaratabelit vaadates võib oletada, et masinad 1 ja 2 on jõudluselt omavahel enam-vähem võrdsed. Seda oletust kinnitasid ka testid 4 ja 5 ning 6 ja 7. Samuti võib riistvaratabeli põhjal oletada, et masin 3 on oma jõudluselt umbes 2 korda võimsam kui masin 1 või 2. Seda oletus leiab kinnitust, kui võrrelda testi 2 tulemust testide 6 või 7 omaga.

Võrreldes testide 2, 6 ja 7 tulemust ning testide 4, 5 ja 8 tulemust leidis kinnitust oletus, et aplikatsiooniserveri ja andmebaasiserveri lahkuviimisel serveri üldine koguvõimsus kasvab. Samas oleks testide tulemuste võrdlemise põhjal eeldanud, et testi 8 tulemus on samas suurusjärgus testide 4 või 5 tulemusega.

Tõenäoliselt võis probleem olla valesti Zope või ZEO serveri vales konfiguratsioonis või siis nendevahelise ühenduse vales konfiguratsioonis, mille tõttu ei koormatud süsteemi 100%-liselt. Samas võib selle põhjuseks olla ka see, et IVA kood ei ole optimeeritud toimimaks mitme masina peal paralleelselt.

Testide tulemusi vaadeldes tekkis küsimus, millal saab ZEO serveri jõudlus klastris probleemiks. Võrreldes nende testide tulemusi, kus aplikatsiooniserver ning andmebaasiserver on eraldatud, võib oletada et iga aplikatsiooniserveri lisamine suurendab andmebaasiserveri koormust ligikaudu 2,5% võrra. Samas tuleb arvestada sellega, et

aplikatsiooniserverid olid oma jõudluselt andmebaasiserverist ligikaudu poole nõrgemad. Eeldusel, et aplikatsiooniserverid ja andmebaasiserver on oma jõudluselt võrdsed, arvan, et ühe aplikatsiooniserveri ning ühe andmebaasiserveri korral on ZEO serveri protsessori koormus suurusjärgus 10-12%. Sellest mõttekäigust lähtuvalt pakun, et võrdsete masinate korral muutub ZEO serveri jõudlus probleemiks siis, kui aplikatsiooniserverite hulk on kasvanud kaheksa kuni kümneni.

4. Kokkuvõte

Nagu käesolev töö näitas, annab aplikatsiooniserveri ja andmebaasiserveri jagamine erinevatesse masinatesse ning paralleelselt mitme aplikatsiooniserveri kasutamine IVA puhul üsna tuntava efekti. Samas ei saa käesolevas töös pakutud lahendust pidada lõplikuks.

Eelkõige tuleb uurida seda, kuidas muudab Zope ja ZEO konfiguratsioonifailide muutmine virtuaalserveri jõudlust. Samuti on vajalik uurida seda, kuidas optimeerida IVA koodi selliselt, et ta toimiks hästi ka klastri peal. Ka tuleb uurida võimalikke virtuaalserveri administreerimiseks ning jälgimiseks mõeldud utiliite ning leida konkreetselt Tallinna Pedagoogikaülikooli jaoks optimaalseim riist- ja tarkvara kooslus.

Kuna teadaolevalt on see Eestis esimene kord, kus Zope veebiserverit püütakse käivitada klastri peal, siis näen käesoleva kirjutise mõtet eelkõige juhendmaterjalina klastri ning sellel Zope serveri toimima saamiseks.

Tahaks loota, et käesolev töö andis mõningaid ideesid, kuidas muuta IVA käideldavust suurte koormuste korral natuke paremaks ning, et ta on abiks ka nende ideede elluviimisel.

5. Kasutatud kirjandus

Mõistete tõlked pärinevad Vello Hanson, Jaan Priisalu Küberneetika Instituudi inglise-prantsuse-soome-eesti arvutisõnastik http://keeks.cyber.ee/arvutisonastik/

[1] IEEE 90

[2] http://www.oracle.com/

[3] http://setiathome.ssl.berkeley.edu/

[4] CORBA http://www.corba.org/

[5] Object Management Group http://www.omg.org/

[6] DCOM http://www.microsoft.com/com/wpaper/default.asp#DCOMpapers

[7] Linux Virtualserver Project http://www.linuxvirtualserver.org/

[8] http://www.oracle.com/database/index.html?rac_home.html

[9] http://www.cisco.com/en/US/products/hw/contnetw/ps792/prod_bulletin09186a008007ca48.html

[10] Clusterknoppix http://bofh.be/clusterknoppix/

[11] MandrakeLinux CLIC http://clic.mandrakesoft.com/index-en.html

[12] Red Hat Enterprise Linux http://www.redhat.com/software/rhel/

[13] OSCAR http://oscar.sf.net/

[14] Cluster SSH http://clusterssh.sourceforge.net/

[15] Kimberlite Webmin moodul http://webmin.IDEALX.org/

[16] OpenMosix http://www.openmosix.org/

[17] OpenMosixView http://www.openmosixview.com/

[18] Ultra Monkey http://www.ultramonkey.org/

[19] TOP500 Superarvutitele pühendatud veebilehekülg http://www.top500.org/

[20] NorduGrid http://www.nordugrid.org/

[21] Eesti Grid http://grid.eenet.ee/

[22] http://www.ietf.org/rfc/rfc1631.txt

[23] http://www.ietf.org/rfc/rfc1853.txt

[24] keepalived http://www.keepalived.org/

[25] http://www.zope.org/Documentation/Books/ZopeBook/

6.Lisa 1 Keepalived konfiguratsioonifail

```
! SEE ON TESTIDES KASUTATUD KOORMUSJAGAJA
! KONFIGURATSIOONFAIL

global_defs {
   notification_email {
     meelis@tpu.ee
     hillarp@tpu.ee
   }
   notification_email_from cluster@tpu.ee
   smtp_server 193.40.239.27
   smtp_connect_timeout 30
   lvs_id test_cluster
}

vrrp_sync_group VG1 {
    GROUP {
      VI_1
      VI_GATEWAY
}

vrrp_instance VI_1 {
    state MASTER
    interface eth0
    virtual_router_id 51
    priority 100
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        192.168.41.246
    }
}

vrrp_instance VI_GATEWAY {
    state MASTER
    interface eth1
    lvs_sync_daemon_interface eth1
    virtual_router_id 51
    priority 100
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        192.168.1.1
    }
}

virtual_server 192.168.41.246 80 {
    delay_loop 6
    lb_algo rr
    lb_kind NAT
    persistence_timeout 50
    protocol TCP
```

```
!    sorry_server 192.168.41.246 90

    real_server 192.168.1.11 8080 {
        weight 1
      HTTP_GET {
          url {
              path /
#           digest b3582fa4b2738526fe194ff39686c715
          }
            connect_timeout 3
          connect_port 8080
            nb_get_retry 3
            delay_before_retry 3
        }
    }

    real_server 192.168.1.12 8080 {
        weight 1
        HTTP_GET {
          url {
              path /
#           digest 1422907eb71b73b5e2667742af954794
          }
            connect_timeout 3
          connect_port 8080
            nb_get_retry 3
            delay_before_retry 3
        }

    real_server 192.168.1.21 8080 {
        weight 1
      HTTP_GET {
         url {
            path /
            digest
         }
         connect_timeout 3
         connect_port 8080
         nb_get_retry 3
         delay_before_retry 3
    }
}
```

7.Lisa 2 ZEO konfiguratsioonifail

```
# ZEO configuration file

%define INSTANCE /var/zope/ZEOinstance

<zeo>
  address 7700
  read-only false
  invalidation-queue-size 100
  # monitor-address PORT
  # transaction-timeout SECONDS
</zeo>

<filestorage 1>
  path $INSTANCE/var/Data.fs
</filestorage>

<eventlog>
  level info
  <logfile>
    path $INSTANCE/log/zeo.log
  </logfile>
</eventlog>

<runner>
  program $INSTANCE/bin/runzeo
  socket-name $INSTANCE/etc/zeo.zdsock
  daemon true
  forever false
  backoff-limit 10
  exit-codes 0, 2
  directory $INSTANCE
  default-to-interactive true
  # user zope
  python /usr/bin/python2.3
  zdrun /var/zope/lib/python/zdaemon/zdrun.py

  logfile $INSTANCE/log/zeo.log
</runner>
```

8.Lisa 3 Zope konfiguratsioonifail

```
### SEE ON PRAKTILISES OSAS KASUTATUD
### ZOPE KONFIGURATSIOONFAIL

%define INSTANCE /var/zope/ZOPEinstantce
%define ZOPE /var/zope

instancehome $INSTANCE

effective-user zope

datetime-format international

<eventlog>
  level all
  <logfile>
    path $INSTANCE/log/event.log
    level info
  </logfile>
</eventlog>

<logger access>
  level WARN
  <logfile>
    path $INSTANCE/log/Z2.log
    format %(message)s
  </logfile>
</logger>

<http-server>
  # valid keys are "address" and "force-connection-close"
  address 8080
  # force-connection-close on
</http-server>

<ftp-server>
  # valid key is "address"
  address 8021
</ftp-server>

<zodb_db temporary>
    # Temporary storage database (for sessions)
    <temporarystorage>
      name temporary storage for sessioning
    </temporarystorage>
    mount-point /temp_folder
    container-class Products.TemporaryFolder.TemporaryContainer
</zodb_db>

 <zodb_db main>
   mount-point /
   <zeoclient>
     server 192.168.1.21:7700
     storage 1
     name zeostorage
     var $INSTANCE/var
   </zeoclient>
 </zodb_db>
```

Summary

Distributed computing in learning systems on example of IVA

IVA is a Web-based learning management system, which is developed in Tallinn Pedagogical University in order to advocate constructively approaches and practices in e-learning. IVA works on Zope web server and his programming language is Python.

Python is a 4'th generation programming language in witch user can give orders to computer in basic English language. Because computer uses inside machine code, that is ones and zeros, it will take time and computers resources to translate human readable language to machine code.

As long there isn't many users, this computing overhead will not remarkable, but when there will be thousands of users, it becomes a problem.

In this case we study how useful is distributed computing in Learning Management Systems such as IVA and how well IVA will work on cluster.

In meaning this work gives an example how about Zope web server and his product IVA can cooperate with cluster. Also, this work aims to give little HOW-TO about setting up cluster and Zope web server with IVA on it.