TAMPEREEN TEKNILLINEN YLIOPISTO

KAI PIRTTIMÄKI

KERNEL-BASED VIRTUAL MACHINEN (KVM) KÄYTTÖ PALVELINVIRTUALISOINNISSA

Diplomityö

Tarkastaja: professori Pekka Loula
Tarkastaja ja aihe hyväksytty Talouden ja rakentamisen tiedekunta-neuvoston kokouksessa 17. elokuuta 2016

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Johtamisen ja tietotekniikan tutkinto-ohjelma
PIRTTIMÄKI, KAI:
Kernel-based Virtual Machinen (KVM) käyttö palvelinvirtualisoinnissa
Server Virtualization Using Kernel-based Virtual Machine (KVM)
Diplomityö, 66 sivua, 2 liitettä
Lokakuu 2016
Pääaine: Tietoverkkojen hallinta ja tietoturva
Tarkastaja: professori Pekka Loula
Avainsanat: Virtualisointi, KVM, avoin lähdekoodi, Linux, Kernel-based virtual machine, palvelinvirtualisointi

Suomi on pienten laitesalien maa. Monella yrityksellä on omat laitesalinsa, joissa ajetaan suurta osaa organisaation tarvitsemasta IT – infrastruktuurista. Usein näiden laitesalien ongelmana on ahtaus, energian kulutus, jäähdytys ja turvallisuus. Ne muodostavat myös kuluerän, jota on vaikea yksilöidä tiettyyn projektiin tai toimintoon kuuluvaksi. Virtualisoinnista on tullut vakioratkaisu moniin näistä ongelmista.

Palvelinvirtualisoinnin pitkät perinteet ulottuvat aina 1960 – luvulle IBM:n keskustietokoneisiin asti. Virtualisointi on tänä aikana kuitenkin muuttunut oleellisesti. Siitä on kasvanut perusta, jonka varaan pilvipalvelut on rakennettu.

Tämä työ käy läpi virtualisoinnin perusteita ja sen monia muotoja, sekä esittelee keskeiset virtualisoinnissa käytettävät tekniset menetelmät. Se kuvaa virtualisoinnin tarjoamia etuja, haasteita ja mahdollisia ongelmakohtia.

Työ esittelee avoimeen lähdekoodin perustuvan Kernel-based Virtual Machine – sovelluspaketin, kuvaa sen virtualisointiytimen arkkitehtuurin ja avaa sen toimintaa. Samalla perehdytään KVM:n käyttämiin tekniikoihin, joilla parannetaan virtuaalikoneiden suorituskykyä ja tehostetaan niiden resurssien käyttöä. Työssä demonstroidaan KVM – virtualisointialustan asennus ja konfigurointi. Samalla kuvataan lukijalle, miten virtuaalikoneet määritellään ja mitä nämä määritykset pitävät sisällään.

Työssä ajetaan joukko mittauksia, jotka kohdistuvat virtuaalikoneissa käytettäviin prosessoreihin, verkkolaitteisiin ja levyohjaimiin. Mittaustulokset analysoidaan ja niitä verrataan muiden julkaisujen vastaaviin tuloksiin.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
Kai Pirttimäki: Server Virtualization Using Kernel-based Virtual Machine (KVM)
Master of Science Thesis, 66 pages, 2 Appendix pages
October 2016
Major: Network Management and Information Security
Examiner: Professor Pekka Loula
Keywords: Virtualization, KVM, open source, Linux, Kernel-based virtual machine, Server Virtualization

Finland is a country of small server rooms. Most companies have their own server rooms, where majority of the required IT – infrastructure is running. Often these server rooms suffer from problems like lack of space, excessive energy consumption, overheating and security. These also form an expenditure that is difficult to divide between different operations or projects. Virtualization has become the standard solution for many of these problems.

Long traditions of server virtualization stretch way back in to the IBM's mainframes of 1960's. There has been many quite substantial changes since then in the field of virtualization. Server virtualization has grown into a foundation upon which cloud services are built.

This thesis goes through the basic theories of virtualization and its many forms. It also presents the fundamental technical methods used for virtualization. This thesis describes the benefits, challenges and potential problems offered by virtualization.

This thesis presents an open-source software package called Kernel-based Virtual Machine and describes the architecture of its hypervisor. It also focuses on the techniques used by KVM to improve virtual machine performance and efficiency of resources consumed. This thesis demonstrates the installation and configuration of Kernel-based Virtual Machine. It also describes to the reader how virtual machines are defined and what do these configurations contain.

This thesis will also show a set of measurements from the processors, network devices and disk controllers used in virtual machines. The results of these measurements are analyzed and compared to the corresponding results of other publications.

ALKUSANAT

Tämä diplomityö on tehty osana Tampereen teknillisen yliopiston Porin laitoksen opintojani. Haluan erityisesti kiittää työni tarkastajia professori Pekka Loulaa ja DI Matti Monnosta saamistani vinkeistä ja palautteesta. Lisäksi haluan välittää vilpittömän kiitoksen Porin laitoksen opintotoimiston henkilökunnalle heidän antamastaan tuesta tutkintoni viimeistelyn kanssa.

SISÄLLYS

1 JOHDANTO....1
2 VIRTUALISOINNIN PERUSTEET....3
2.1 Virtualisoinnin historiaa....3
2.2 Mitä virtualisointi tarkoittaa....4
2.3 Virtualisoinnin käyttökohteet....5
2.4 Tärkeimmät virtualisointiin liittyvät luokittelut....7
2.4.1 Täysvirtualisointi....8
2.4.2 Paravirtualisointi....8
2.4.3 Laitteistoavusteinen virtualisointi....9
2.4.4 Käyttöjärjestelmätasolla tapahtuva virtualisointi....10
2.4.5 Virtualisointiytimen luokittelu....11
2.5 Virtualisoinnit edut....12
2.6 Virtualisoinnin haasteet, haitat ja potentiaaliset ongelmakohdat....15
2.7 CPU:n Ring-levelit ja x86-tekniikan erityispiirteet....16
2.8 SR-IOV....19
3 KERNEL-BASED VIRTUAL MACHINE....21
3.1 Sovelluspinosta koostuva virtualisointiohjelmisto....21
3.1.1 KVM, Linux-ytimen moduuli....22
3.1.2 Qemu-kvm....23
3.2 KVM:n toiminta....24
3.3 Muistinhallinta....26
3.3.1 KSM – Kernel Same-page Merging....26
3.3.2 Memory Balloon driver....29
3.4 Virtuaalikoneiden tietoliikenne....30
3.5 Tallennusratkaisut....34
4 KVM:N KÄYTTÖÖNOTTO....36
4.1 Virtualisointialustan laitevaatimukset....36
4.2 Hyödylliset ja tarvittavat aputyökalut....36
4.2.1 Libvirt ja virsh....37
4.2.2 Bridge-utils, OpenSSH-server sekä valvontatyökalut....39
4.2.3 Virtio....40
4.3 Virtualisointialustan asennus ja konfigurointi....41
4.4 Virtuaalikoneen määrittely ja käyttöönotto....42
5 MITTAUSTULOKSET....46
5.1 Prosessorin suorituskyky....46
5.2 Verkkoliikenteen suorituskyky....48
5.3 Levyohjaimien suorituskyky....51
5.4 Mittausten vertailu muiden julkaisemiin tuloksiin....54

6 YHTEENVETO.....60
LÄHTEET.....62

Liite A: KVM-Qemu, prosessorituki ja tunnistetut CPU-ominaisuudet
Liite B: KVM API, yksinkertaisen virtuaalikoneen luominen 16-bittisen x86-koodin ajamista varten.

Kuvaluettelo

Kuva 2.1: Kuvassa virtualisointiytimen tyypit 1 ja 2. Mukaillen (Milberg et al. 2013)..11

Kuva 2.2: Virtualisoinnin edut, laiteabstrahointi.....14

Kuva 2.3: Kuvassa x86 - tekniikan ring levelit. Vasemmalla perinteinen malli, oikealla täysvirtualisoinnissa käytetty malli. Mukaillen (Red Hat 2015).....17

Kuva 2.4: SR-IOV - tekniikka mahdollistaa yhden PCIe - laitteen jaon useammalle virtuaalikoneelle. (Herrmann et al. 2016).....19

Kuva 3.1: KVM-arkkitehtuuri. Linux-ydin ja KVM-moduuli muodostavat yhdessä virtualisointiytimen, joka tarjoaa palveluja ylemmille kerroksille.....22

Kuva 3.2: Silmukka, jossa virtuaalikonetta suoritetaan. Kuva mukaillen (Kivity et al. 2007).....25

Kuva 3.3: KSM: alkutilanne ennen muistisivujen yhdistämistä. (Guangrong 2011).…27

Kuva 3.4: KSM: Tilanne muistisivujen yhdistämisen jälkeen. Muistialue on merkitty kirjoitettaessa kopioitavaksi. (Guangrong 2011).....27

Kuva 3.5: KSM: Tilanne muistialueen sisällön muuttamisen jälkeen. (Guangrong 2011) 28

Kuva 3.6: Kaaviossa on esitetty Virtual memory balloon - ajurin toiminta yhteistyössä virtualisointiytimen ja virtuaalikoneen käyttöjärjestelmäytimen kanssa.....29

Kuva 3.7: KVM:n verkkoliikenteen I/O - arkkitehtuuri (Li et al. 2009).....31

Kuva 3.8: Verkkoarkkitehtuurin nopeuttamiseksi ehdotettu vaihtoehto, jossa virtualisointialustan kernel modessa toimiva säie vastaanottaa saapuvat paketit ja kopioi ne suoraan virtuaaliverkkokortille. (Li et al. 2009).....32

Kuva 3.9: Vhost-net - modulin toiminta. Verkkoliikenteen käsittely on siirretty pois user spacesta toimivalta QEMU - prosessilta ja siirretty virtualisointiytimen vhost – moduliin. (Burns 2010).....33

Kuva 4.1: Libvirt - ohjelmiston tukemat virtualisointijärjestelmät ja asiakasohjelmat (Ashley et al. 2016).....37

Kuva 4.2: Libvirt - hallintatyökalujen protokollapino. Kuvassa Linux käsittää KVM:n tapauksessa KVM - moduulin ja Linux - ytimen muodostaman virtualisointialustan. (Jones 2012).....39

Kuva 4.3: Virtio - ajuripaketin komponettien rakenne. (Jones 2010).....40

Kuva 4.4: Kaikki Virtio:n tarjoamat laiteajurit. (Jones 2010).....41

Kuva 5.1: Sysbench-testiä käytettiin selvittämään suorituskykyeroa virtualisoidun prosessorin ja alustan prosessorin välillä, ja eri konfiguraatioiden vaikutusta tuloksiin. Pienempi aika parempi. Ajan alla myös suhdeluku alustan aikaan verrattuna.....47

Kuva 5.2: Verkkomittauksissa käytetty testikokoonpano. Kuvassa eritelty emuloidun ja paravirtualisoidun verkkoarkkitehtuurin erot. Virtuaalikoneet ja testikone on kytketty HP:n kytkimen kautta toisiinsa yhden gigabitin verkkolinkillä. Kuva mukaillen (Li et al.

2009) ja (Burns 2010).....48

Kuva 5.3: Verkon nopeustestit iperf - ohjelmistolla. Mitattuna liikenne palvelimelta verkkoon (output) ja verkosta palvelimelle (input).....49

Kuva 5.4: Verkon nopeustestit netcat- ja dd - ohjelmistolla. Mitattuna liikenne palvelimelta verkkoon (output) ja verkosta palvelimelle (input).....50

Kuva 5.5: Sysbenchin fileio - testin tulokset. Tulokset mittaavat levyjärjestelmän suorituskykyä erikokoisilla tiedostoilla.....51

Kuva 5.6: Levyn kirjoitusnopeus testattuna dd - ohjelmalla, yksi 10GB tiedosto.....52

Kuva 5.7: Kirjoitusnopeustesti, virtuaalikoneen ja virtualisointialustan suorituskyky ja CPU – kuorma kirjoitettavan lohkon koon funktiona. (Zhang et al. 2010).....52

Kuva 5.8: Lukunopeustesti, virtuaalikoneen ja virtualisointialustan suorituskyky ja CPU – kuorma luettavan lohkon koon funktiona.(Zhang et al. 2010).....53

Kuva 5.9: HEP-Spec06 - testiohjelman ajoa eri määrässä virtuaalikoneita ja verrattuna prosessorin natiiviin suorituskykyyn. Xeon E5420 – prosessorissa on neljä fyysistä prosessoriydintä, jotka tukevat hyperthreading – tekniikkaa. Tämä selittää eron suorituskyvyssä neljää useammalla virtuaalikoneella. (Chierici et al. 2010).....55

Kuva 5.10: Emuloidun E1000 – verkkokortin nopeusvertailua virtualisointialustaan nähden iperf – ohjelmalla. Vasta kahdeksan testiä ajavaa virtuaalikonetta saivat fyysisen linkin täyteen. (Chierici et al. 2010).....56

Kuva 5.11: Netperf - mittaustulokset. Vertailu emuloitujen ja fyysisen verkkokortin nopeuksista. (Zhang et al. 2010).....57

Kuva 5.12: Levykuvatiedostoformaattien nopeusvertailu, kyseessä lukunopeustesti (Kourai et al. 2014).....58

Kuva 5.13: Levy-I/O - nopeustestien tulokset. KVM:n ja Xenin päällä ajetaan yhtä virtuaalikonetta, vertailun vuoksi virtualisointialustan tulokset. (Soriga et al. 2013)…59

TERMIT JA NIIDEN MÄÄRITELMÄT

/dev/null	Unix – tyyppisissä käyttöjärjestelmissä oleva tiedosto, jonne syötetty data ei ohjaudu mihinkään. Tyypillisesti sitä käytetään kun halutaan piilottaa prosessien tarpeeton tuloste näkyviltä.
Allokoida	Kohdentaa, varata. Esimerkiksi käyttöjärjestelmä voi varata tietyn muistialueen prosessia varten.
BYOD	Bring Your Own Device. Käytäntö joka sallii työntekijöiden käyttää omia henkilökohtaisia päätelaitteitaan yrityksen verkossa työntekemiseen.
DAS	Direct Attached Storage. Suorakytkentäiset tallennusjärjestelmät (DAS) ovat nimensä mukaisesti kytkettynä suoraan käytettävään tietokonejärjestelmään.
Emulointi	Toisen laitteen tai järjestelmän toiminnan kattavaa jäljittelyä. Virtualisoinnin yhteydessä tarkoitetaan yleensä olemassa olevan fyysisen laitteen toiminnot toteuttavaa ohjelmistokomponenttia. Emulointi on simuloinnin osajoukko.
FTP	File Transfer Protocol on tietoverkoissa käytettävä salaamaton tiedonsiirtoprotokolla.
IaaS	Infrastructure as a Service tarkoittaa IT-infran, eli esimerkiksi palvelimien, tallennustilan tai verkkolaitteiden tuottamista asiakkaan käyttöön.
iSCSI	Internet Small Computer Systems interface. IP – protokollan päällä SCSI – käskyjä siirtävä tiedonsiirtoprotokolla. Sen tarkoituksena on tarjota lohkotason yhteys verkon yli käytettävään tallennuslaitteeseen.
KVM	Kernel-based Virtual Machine. Avoimen lähdekoodin ohjelmistopaketti, joka mahdollistaa usean virtuaalisen tietokoneinstanssin ja käyttöjärjestelmän ajon samanaikaisesti yhdellä fyysisellä laitteistolla.
Käyttäjätila	User Space. Linux-ytimen toimitila, joka vastaa karkeasti x86-arkkitehtuurin rajoitetuinta ring leveliä 3. Kyseisessä tilassa suoritetaan normaalien prosessien ajaminen.
Laitteistoavusteinen virtualisointi	Laitteistoavusteinen virtualisointi on ensimmäisen kerran vuonna 2005 esiteltyjen x86 - arkkitehtuurin virtualisointilaajennusten jälkeen muodostunut ylivoimaisesti käytetyimmäksi virtualisointimenetelmäksi. Se

	perustuu ohjelmistopohjaisen täysvirtualisoinnin tavoin virtualisointiytimen tekemään laite-emulointiin, mutta prosessorien virtualisointilaajennusten hyödyntäminen siirtää suuren osan emulaatiota ohjelmistotasolta raudan hoidettavaksi.
Lohkotaso	Block Level. Lohkotason tallennusjärjestelmissä data on jaettu lohkoihin kuten perinteisissä lohkolaitteissa (esim. kiintolevyt). Tallennusvirtualisoinnis yleistyessä termi on laajentunut käsittämään tallennusmenetelmät, jotka simuloivat lohkotason laitteen toimintaa.
NAS	Network Attached Storage. Verkkokytkentäiset tallennusjärjestelmät NAS ovat tyypillisesti TCP/IP – protokollaa hyödyntävien verkkojen yli käytettäviä laitteita, jotka tarjoavat paikalliselle tietokoneelle tallennustilaa. Tiedonsiirtoon niiden kanssa käytetään esimerkiksi NFS-, SMB- tai iSCSI – protokollia.
NFS	Network File System on protokolla, jolla jaetaan tiedostoja verkon yli palvelimelta asiakaskoneelle.
PaaS	Platform as a Service tarkoittaa palvelualustojen tuottamista asiakkaiden käyttöön. Tämä käsittää esimerkiksi tietokanta-, WWW- tai sovellusalustojen tuottamista asiakkaan käyttöön.
Paravirtualisointi	Paravirtualization. Virtualisointimenetelmä, jossa virtuaalikoneessa käytettävää käyttöjärjestelmäydintä on muutettu niin, että aiemmin laitteistolle tarkoitetut operaatiot ohjataan virtualisointiytimen käsiteltäväksi.
SaaS	Software as a Service tarkoittaa ohjelmiston tuottamista palveluna. Tyypillisesti palveluntarjoaja vastaa sekä ohjelmiston että alempien kerrosten ylläpitämisestä ja asiakas voi keskittyvä vain ohjelmiston käyttöön.
SAN	Storage area networks. SAN-järjestelmät ovat tyypillisesti valokuituyhteyksien yli käytettäviä tallennuslaitteita, jotka käyttävät FCP-, iSCSI- tai HyperSCSI – protokollia tiedonsiirtoon.
Simulointi	Simulaatio on järjestelmä, joka ulkoisesti vaikuttaa toimivan kuten kohdejärjestelmä, mutta on voitu sisäisesti toteuttaa täysin erityyppisellä ratkaisulla.
SMB	Server Message Block. Tiedonsiirtoprotokolla, jolla jaetaan tiedostoja verkon yli palvelimelta asiakaskoneelle.
SPICE	Simple Protocol for Independent Computing Environments. Etäyhteysprotokolla, jota käytetään

	virtuaalikoneiden näkymän ja käyttäjän antaman syötteen välittämiseen.
SSH	Secure Shell. SSH – protokolla on salattu tietoverkoissa käytettävä tiedonsiirtoprotokolla.
stderr	Linux standard error. Prosessin tulostamat virheilmoitukset ohjataan stderrin kautta eteenpäin.
stdin	Linux standard input. Aiemman prosessin tuloste voidaan putkittaa stdinin kautta syötteeksi toiselle prosessille.
stdout	Linux standard output. Linuxissa prosessien tuloste ohjataan stdoutin kautta tyypillisesti käyttäjän nähtäville komentoriville. Tuloste voidaan ohjata myös esimerkiksi tiedostoon, verkkosokettin, sarjaporttiin tai syötteeksi toiselle ohjelmalle.
Thin Client	Pääte. Yksinkertainen tietokone, joka pääsääntöisesti on suunniteltu keräämään syötettä käyttäjältä ja näyttämään tulostetta käyttäjälle. Päätteitä käytetään palvelimien etäkäyttöön.
Täysvirtualisointi	Täysvirtualisoinnissa virtualisointiohjelmisto tarjoaa virtuaalikoneelle emuloidun laiteympäristön, jonka päällä vieraskäyttöjärjestelmää voidaan ajaa. Yleensä täysvirtualisoinnilla viitataan nykään nimenomaisesti ohjelmistopohjaiseen täysvirtualisointiin erotuksena laitteistoavusteisesta virtualisoinnista.
Vierastila	Guest Mode. Prosessorivalmistajien VT-X ja AMD-V – laajennusten myötä käyttöönottama prosessorin toimitila, joka on tarkoitettu mahdollistamaan laitteistoavusteinen virtualisointi. Vierastilasta poistuttaessa prosessorin hallinta palautetaan virtualisointiytimelle.
Virtualisointiydin	Hypervisor. Ohjelmistokomponentti, joka tarjoaa ja hallinnoi käytettävän laitteiston palveluja virtuaalikoneille ja muille ohjelmistoprosesseille.
Vuorontaja	Scheduler. Ohjelmistokomponentti, joka moniajoympäristöissä jakaa laitteistoresursseja ylemmille ohjelmistokerroksille.
WHQL-sertifioitu laiteajuri	Windows Hardware Quality Labs. WHQL-ajurit ovat Microsoftin Windows-käyttöjärjestelmää varten sertifioimia laiteajureita.
Ydintila	Linux-ytimen toimitila, joka vastaa mm. x86-arkkitehtuurin etuoikeutettua tilaa. (engl. kernel mode)

1 JOHDANTO

Suomi on pienten laitesalien maa. Monella yrityksellä on omat laitesalinsa, joissa pyöritetään suurta osaa organisaation tarvitsemasta IT – infrastruktuurista. Usein näiden laitesalien ongelmana on ahtaus, energian kulutus, jäähdytys ja turvallisuus. Ne ovat täynnä palvelimia, jotka muodostavat kuluerän jota on vaikea yksilöidä tiettyyn projektiin tai toimintoon kuuluvaksi.

Normaalin, virtualisoimattoman palvelimen kuorma on tavallisesti noin 5% - 15% luokkaa (Meier 2008). Koska palvelimen laitteisto ja ennen kaikkea sähkönsyöttö on mitoitettu maksimaalisen tehontarpeen mukaan, ja on usein vielä kahdennettu, syntyy huomattava määrä hukkalämpöä varsinaiseen hyötykäytettyyn prosessorointitehoon nähden. Merkittävä osa tätä ongelmaa on palvelimien virtalähteet, jotka on suunniteltu tarjoamaan hyvän hyötysuhteen verrattain lähellä nimellistehoaan, eikä suinkaan matalalla kuormalla. Ja kuten fysiikasta tunnetaan, energia ei ole katoavaista, joten tämä lämpö täytyy suurelta osin siirtää laitehuoneen ulkopuolelle jäähdytyksen avustuksella. Virtualisointi mahdollistaa palvelimen kuorman nostamisen merkittävästi korkeammaksi, suositusten mukaan jopa 80% asti, ajamalla fyysisellä palvelinraudalla useampia virtuaalikoneita (IBM 2012). Tämä kasvattaa hyötykäytetyn prosessorointitehon osuutta käytetyn sähköenergian määrään nähden ja samalla vie fyysisesti vähemmän tilaa ja aiheuttaa täten vähemmän laitekuluja.

Virtualisointi helpottaa myös palvelimien hallintaa. Kun yksi palvelu tai yksi järkevä palvelukokonaisuus keskitetään yhdelle virtuaalikoneelle, vähenevät riippuvuussuhteet muihin palvelimiin merkittävästi. Tämä yksinkertaistaa palvelimien hallintaa ja ylläpitoa, sillä yhden kokonaisuuden muodostavan mustan laatikon rajapinnat ulkomaailmaan on helppo dokumentoida. Se helpottaa palvelimien elinkaarten hallintaa: yksi kokonainen musta laatikko voidaan korvata toisella kun sellainen tarve tulee, kunhan huolehditaan rajapinnan tarjoamien palveluiden toteutuksesta. Myös ylläpito on helpompaa, kun ei tarvitse huolehtia riippuvuussuhteista muihin palveluihin. Ja kun palvelimien rajapinnat ja käyttötarkoitukset tunnetaan hyvin, voidaan niiden kulut helpommin kohdistaa oikealle momentille.

Tämä diplomityö selvittää lukijalle palvelinvirtualisoinnissa tarvittavia tekniikoita, niiden taustoja, toimintaa sekä niihin liittyviä etuja ja mahdollisia haittoja. Työssä esitellään avoimeen lähdekoodiin perustuva Kernel-based Virtual Machine – virtualisointijärjestelmä, kuvataan sen arkkitehtuuri ja avataan sen toimintaa. Samalla perehdytään KVM:n käyttämiin tekniikoihin, joilla parannetaan virtuaalikoneiden suorituskykyä ja tehostetaan niiden resurssien käyttöä. Työssä myös demonstroidaan KVM – virtuali-

sointialustan asennus ja konfigurointi. Samalla kuvataan lukijalle, miten virtuaalikoneet määritellään ja mitä nämä määritykset pitävät sisällään.

Työ on jaettu kuuteen eri lukuun. Luvussa kaksi käydään läpi yleinen virtualisointiin liittyvä teoria. Luku kolme keskittyy KVM – virtualisointialustan toimintaan ja luku neljä puolestaan sen käyttöönottoon.

Viidennessä luvussa tutkitaan KVM:n suorituskykyä mittausten avulla ja ajetaan joukko mittauksia, jotka kohdistuvat virtuaalikoneissa käytettäviin prosessoreihin, verkkolaitteisiin ja levyohjaimiin. Nämä mittaukset pyritään kytkemään niihin liittyvään teoriapohjaan ja samalla analysoidaan miten mittaustulokset vaikuttavat virtualisointiin käytännössä. Mittaustuloksia verrataan myös muiden julkaisujen vastaaviin tuloksiin ja tehdään sitä kautta johtopäätöksiä mittaustulosten kattavuudesta. Työ päättyy luvun 6 yhteenvetoon, joka kokoaa kasaan työn kannalta merkittävimmät havainnot ja tulokset.

2 VIRTUALISOINNIN PERUSTEET

Luvussa käydään lyhyesti läpi virtualisoinnin historiaa sekä avataan virtualisointiin liittyviä käsitteitä ja erilaisia virtualisointitapoja. Lisäksi käsitellään x86 – arkkitehtuurin suojaustasoja ja niiden vaikutusta virtualisointiin sekä käydään läpi virtualisoinnin tarjoamia etuja ja mahdollisia haittoja.

2.1 Virtualisoinnin historiaa

Tietotekniikkaresurssien virtualisointi ei ole alalla uusi idea, vaikka sen sisältö onkin ajan myötä muuttunut todella paljon. IBM käynnisti uranuurtavan työn virtualisointiin liittyen jo vuonna 1964, kun Robert Creasy ja Les Comeau aloittivat aiheen tutkimisen System/360 – keskustietokoneen ja CP-40 – käyttöjärjestelmän parissa. Tämä ratkaisu jäi kuitenkin lähinnä laboratoriokäyttöön. Ensimmäinen kaupalliseksi tuotteeksi asti yltänyt projekti sai alkunsa 1968, kun Jim Rymarczyk aloitti IBM:n virtualisointiytimen toisen version kehitystyön. Tämä johti edellisen keskustietokonesukupolven kanssa yhteensopivaan System/360 model 67 – keskustietokoneen sekä CP/CMS käyttöjärjestelmän hallintaohjelman version CP-67 julkaisuun. (Bradkin 2009)

CP-67:n kantavana ideana oli yksinkertaistaa CP/CMS – käyttöjärjestelmän kehitystyötä eriyttämällä laitteistoresurssien hallinnointi omaksi komponentikseen tarjoten abstraktiotason raudan ja ylempien sovelluskerrosten välille. Resurssien jako oli aikaperusteista, joten käytännössä CP toimi käyttöjärjestelmän vuorontajana (engl. scheduler) mahdollistaen moniajon eli usean ohjelman ajamisen näennäisesti samaan aikaan. (Bradkin 2009)

Keskustietokoneen jokaiselle käyttäjälle tarjottiin oma CMS (Conversational Monitor System), joka oli yhden käyttäjän käyttöjärjestelmä. CMS:n ei keskustellut suoraan fyysisen laitteiston kanssa vaan se käytti CP:n tarjoamia virtuaaliresursseja. Tämä suoraviivaisti CMS:n kehitystyötä ja samalla mahdollisti jo aiemmin kirjoitettujen ohjelmistojen käytön koko fyysistä laitteistoa varaamatta. (Bradkin 2009)

VMWare kehitti vuonna 1998 menetelmän, joka mahdollisti virtualisoinnin sen aikaisilla x86-prosessoreilla. Tämä menetelmä johti vuonna 1999 VMWare Workstation – tuotteen julkaisemiseen. Sen ensimmäiset versiot tarjosivat virtualisointia 32-bittisillä x86-prosessoreilla alustanaan Linux tai Microsoft Windows. Pian tämän jälkeen VMWare julkaisi ESX Server-tuotesarjansa, jonka ytimenä toimi yhtiön oma VMKernel. Koska x86 - arkkitehtuuri ei vielä tarjonnut rautapohjaista mahdollisuutta virtuali-

sointiin, käytti VMWare omaa ohjelmistopohjaista ratkaisuaan, BT - tekniikkaa (Binary Translation). Tekniikan kantavana ajatuksena on se, että VMKernel analysoi virtuaalikoneiden muistista löytyvät x86 - arkkitehtuurin käskyt ennen niiden suoritusta ja korvasi laitteistoa rajoittamattomasti käsittelevät käskyt sarjalla uusia käskyjä, jotka oli turvallista suorittaa ilman että virtuaalikoneiden suorittamat operaatiot haittaisivat toisiaan. (VMWare 2007, VMWare 2009)

Samoihin aikoihin VMWaren tuotejulkistusten kanssa Keir Fraser ja Ian Pratts aloittivat Cambridgen yliopistossa osana Xenoserver - projektia Xen - virtualisointiytimen kehityksen. Projektin lähdekoodi avattiin vuonna 2002, ja jo seuraavana vuonna julkaistiin ensimmäinen julkinen versio Xenistä. Ohjelmisto käytti hyväkseen paravirtualisointia, jossa ajettavan virtuaalikoneen ydintä pitää muuttaa niin, että se osaa hyödyntää virtualisointialustan tarjoama ohjelmointirajapintaa. Tämä rajoitti merkittävästi erityisesti suljetun lähdekoodin käyttöjärjestelmien käyttöä virtuaalikoneina, mutta toisaalta nopeutti virtualisointi merkittävästi. Paravirtualisointi onkin osatekniikkana käytössä monessa nykyaikaisessa virtualisointijärjestelmässä, ja sitä käytetäänkin erityisesti virtuaalisten laiteajurien kanssa. Asiaa käsitellään tarkemmin myöhemmässä luvussa. (Xen 2016)

Virtualisointi x86-pohjaisella prosessoriarkkitehtuurilla muuttui ratkaisevasti, kun kaksi merkittävintä suoritinvalmistajaa laajensivat kumpikin omilla tahoillaan x86-käskykantaa uusilla virtualisointia tukevilla laajennuksilla. Ensimmäiset Intelin VT-x – laajennusta tukevat prosessorit tulivat markkinoille marraskuussa 2005 ja AMD:n AMD-V – laajennuksella varustetut prosessorit toukokuussa 2006. Laajennukset ovat keskenään erilaisia, mutta käyttävät hyvin samankaltaista lähestymistapaa virtualisointiin. Käytännössä ne lisäävät prosessorille kyvyn olla joko isäntä- (engl. host mode) tai vierastilassa (engl. guest mode). Tämä avasi mahdollisuuden uudenlaiseen ja helpompaan x86-virtualisointiin, sillä laitteiston tarjoamia ominaisuuksia ei enää tarvinnut toteuttaa virtualisointiytimien ohjelmakoodissa. (Red Hat 2015)

Tämä tekninen kehitys johti virtualisointijärjestelmien muuttumiseen. Erityisesti open source – kentälle ilmestyi myös sen myötä uusia toimijoita. Yksi näistä oli Qumranet – yhtiön julkaisema Kernel-based Virtual Machine, KVM. (Burns 2010)

Melko pian julkaisunsa jälkeen KVM sai open source – maailmassa standardin aseman, sillä se hyväksyttiin Linux-ytimen osaksi vuonna 2007. Tämä asema vahvistui vielä entisestään, kun painoarvoltaan merkittävimpiin Linux-jakelijoihin kuuluva Red Hat osti KVM:n vuonna 2008 ja päätti tarjota sille virallista tukea vuoden 2009 Red Hat Enterprise Linux 5.4 – versiosta lähtien. (Burns 2010)

2.2 Mitä virtualisointi tarkoittaa

Perinteisesti tietokoneella on ajettu kerrallaan vain yhtä sovellusohjelmistoa, jolle käyttöjärjestelmä on tarjonnut liitynnät laitteiston resursseihin. Ohjelmistot sovitettiin toimi-

vaksi tietyn käyttöjärjestelmän ja laitteiston kanssa, ja aina kun nämä muuttuivat oleellisesti, ohjelmistotoimittajan piti uudelleenkirjoittaa osa ohjelmaa vastaamaan uusia olosuhteita. Vähitellen tämän työn minimoimiseksi kehittyi yleisesti sovittuja rajapintoja, abstraktiotasoja, jotka helpottivat sovitustyötä (Geer 2009).

Virtualisointi on vastaavanlainen abstraktiotaso. Virtualisointitavasta riippuen se voi käsittää esimerkiksi kokonaisten tietokonelaitteistojen emuloimista käyttöjärjestelmä- ja ohjelmistokerrokselle niin, ettei niiden käyttäjä edes välttämättä ole tietoinen virtualisoinnista. Toisaalta, virtualisointia tapahtuu myös paljon pienempien rajapintojen tasolla; esimerkiksi nykyaikaiset käyttöjärjestelmät antavat sovellusohjelmien käyttöön suoran keskusmuistiosoituksen sijaan virtuaalisen muistialueen, jonka sisältö voi todellisuudessa sijaita hajautettuna ympäri keskusmuistin muistiavaruutta tai jopa massamuistilla. Samalla tämä tarkoittaa sitä, että käyttöjärjestelmä voi vapaasti priorisoida osoitettavissa olevan muistikapasiteetin käyttöä tarpeiden mukaan ja sovellusohjelmien toimintaa mitenkään häiritsemättä. (Stallings 2012)

Virtualisointitekniikka mahdollistaa paitsi käytettävissä olevien resurssien abstrahoinnin, myös niiden jakamisen useiden järjestelmien kesken. Siinä missä käyttöjärjestelmätason moniajo jakaa laiteresurssiaikaa ajettavien prosessien kesken, virtualisointi tekee saman asian käyttöjärjestelmän ja laitekerroksen välisellä tasolla. Tyypillisesti virtualisoinnissa halutaan myös varmistaa, että virtualisoidut järjestelmät ovat kapsuloituja omaan kuplaansa, eivätkä pysty hallitsemattomasti vaikuttamaan toistensa tai alla olevan laitteiston toimintaan. Käytännössä tätä hyödynnetään muun muassa kun halutaan myydä palveluna virtualisoituja järjestelmiä asiakkaille niin sanottujen pilvipalveluiden kautta. Asiakkaat voivat ostaa toimittajalta palvelua usealla eri tasolla (IaaS, PaaS, SaaS), eivätkä eri asiakkaiden virtuaalipalvelut ole tekemisissä toistensa kanssa, vaikka niitä samalla virtualisointijärjestelmällä ajettaisiinkin. (Soriga et al. 2013)

2.3 Virtualisoinnin käyttökohteet

Virtualisointia käytetään nykyään hyvin monella eri osa-alueella. Pitkäperinteisen, aina 60 – luvun puolivälistä kumpuavan palvelinraudan virtualisoinnin rinnalle on ilmestynyt aivan uusia tapoja hyödyntää virtualisoinnin tarjoamia mahdollisuuksia (Bradkin 2009). Palvelimissa käytettävät virtualisointimenetelmät ovat lisäksi muuttuneet suuresti. Näitä muutoksia ja etuja käsitellään myöhemmässä luvussa. (Ruest et al. 2009)

Työpöytävirtualisointi on yleisnimitys menetelmille, jotka mahdollistavat käyttäjältä kerättävän I/O:n (näppäimistö, hiiri) tai hänelle välitettävän I/O:n (monitorin kuva, audio) erottamisen varsinaisesta tietokoneesta, jossa laskenta ja datan käsittely tapahtuu. Tämänkaltaista erottamista on tehty jo aikoinaan keskustietokoneiden aikakaudella, jolloin niin sanotut tyhmät päätteet (tai thin clientit) tarjosivat käyttäjille käyttöympäristön keskustietokoneille. Nykyään esimerkkinä voidaan käyttää RDP-, VNC- tai X-ikkunointijärjestelmän etäkäytön tarjoamia etätyöpöytäyhteyksiä, jotka toimivat asiakas-palvelin

– mallin mukaisesti tarjoten rajapinnan palvelimen ja asiakaspäätteen välille. Myös monissa pilvipalveluissa on vahvasti mukana työpöytävirtualisoinnin piirteitä. SaaS – tyyppiset ratkaisut, joissa asiakas voi esimerkiksi internet-selaimella käyttää palveluntarjoajan palvelimilla toimivia ohjelmistoja (kuten esimerkiksi Google Docs) ovat saavuttaneet melkoisen suosion. (Bowker 2016)

Etätyöpöytäyhteydet tarjotaan tyypillisesti palvelimilta, jotka lähtökohtaisesti on suunniteltu tarjoamaan usealle käyttäjälle pääsyn samaan käyttöjärjestelmäinstanssiin ja sen tarjoamiin palveluihin. Työpöytävirtualisointi voidaan toteuttaa myös menetelmällä, jossa tarpeen mukaan käyttäjälle luodaan joko kokonaan oma käyttöjärjestelmäinstanssi ohjelmistoineen tai pelkkä ohjelmistoinstanssi. Tämäntyyppisiä menetelmiä kutsutaan VDI:ksi (engl. Virtual Desktop Infrastructure). VDI – palvelua tarjoavat ohjelmistot, joista esimerkkinä mainittakoon VMWaren Horizon, nojautuvat vahvasti virtualisointiin. (Ruest et al. 2009, Bowker 2016)

VDI – ohjelmistojen avulla yritysten IT – osastot voivat helposti julkaista käyttäjille kokonaisia virtualisoituja käyttöjärjestelmäinstansseja, jotka luodaan sillä hetkellä kun käyttäjä palvelua tarvitsee. Instanssit kloonataan etukäteen tehdyistä levykuvista ja asetustiedostoista, ja ne ajetaan palvelimien virtualisointiympäristöissä. Kaikki ajonaikaiset muutokset järjestelmiin tallentuvat erilleen levykuvista. Kun instanssin ajo lopetetaan, voidaan muutokset tuhota. Tekniikka vähentää merkittävästi työasemien ylläpitoon tarvittavia resursseja, sillä ylläpidettävänä on vain vakioidut levykuvat, joista kaikki käyttäjien virtuaalikoneet luodaan. Tyypillisesti käyttäjillä on yksinkertaiset ja pitkäikäiset thin clientit käytössään, tai he käyttävät VDI – palveluja BYOD – hengessä omilta henkilökohtaisilta päätelaitteiltaan. (Bowker 2016)

Virtualisointi on tuonut mukanaan muutoksia myös tallennusjärjestelmiin ja siihen, mihin tietoa tallennetaan. Tallennusjärjestelmien virtualisoitumisen ajatellaan yleensä alkaneen tietokoneiden verkottumisen myötä: kaikkea käytettävää dataa ei ole ollut tarpeen tai edes järkevää säilyttää paikallisesti tietokoneen massamuistivälineillä, vaan data on voinut olla hajautettuna useisiin eri järjestelmiin. Järjestelmien välillä tapahtuvaan datansiirtoon on kehitetty hyvin monentyyppisiä protokollia, kuten esimerkiksi FTP, SMB, NFS, iSCSI tai FCP (Fibre Channel Protocol). Virtualisointi on myös tarkoittanut samalla sitä, että tallennusjärjestelmien fyysisiä massamuisteja on yhdistetty uudenlaisiksi loogisiksi järjestelmiksi, jotka tarjoavat vikasietoisuutta, nopeutta, laajennettavuutta ja tallennetun datan deduplikointia ihan eri mittakaavassa kuin mitä yksittäisiltä massamuistilaitteilta on totuttu odottamaan. (Meier 2008)

Tallennusjärjestelmät luokitellaan tyypillisesti kolmeen eri kategoriaan liitäntätapansa mukaisesti: Suorakytkentäiset tallennusjärjestelmät DAS (engl. direct attached storage) ovat nimensä mukaisesti kytkettynä suoraan käytettävään tietokonejärjestelmään. Niiden kanssa käytettyjä protokollia ovat muun muassa SCSI, SAS, SATA, ATA, USB ja Fibre channel. Verkkokytkentäiset tallennusjärjestelmät NAS (engl. Network attached storage) ovat tyypillisesti TCP/IP – protokollaa hyödyntävien verkkojen yli käy-

tettäviä laitteita, jotka tarjoavat paikalliselle tietokoneelle tallennustilaa. Tiedonsiirtoon niiden kanssa käytetään esimerkiksi NFS-, SMB- tai iSCSI – protokollia. SAN-järjestelmät (engl. Storage area networks) ovat tyypillisesti valokuituyhteyksien yli käytettäviä tallennuslaitteita, jotka käyttävät FCP-, iSCSI- tai HyperSCSI – protokollia tiedonsiirtoon. (Ruest et al. 2009, Stallings 2012)

Nämä kategoriat voidaan vielä lajitella kahteen eri ryhmään, lohkotason- (engl. Block level) ja tiedostotason (engl. File level) tallennusjärjestelmiin käytetyn liitännän ja protokollan mukaisesti. DAS- ja SAN - järjestelmät ovat lohkotason tallennusjärjestelmiä, mutta myös iSCSI – protokollaa käyttävät NAS – laitteet kuuluvat niihin. Tiedostotason tallennusjärjestelmiin kuuluvat muun muassa NFS- ja SMB-protokollia käyttävät NAS – laitteet. Luokituksen ovat virtualisoinnin kannalta oleellisia, sillä lohkotason virtualisointimenetelmissä yksittäisen tallennuslohkon looginen ja fyysinen sijainti on erotettu toisistaan abstraktiotasolla. Tyypillisesti tämä abstrahointi toteutetaan Address space remappingillä, joka vastaa osoitemuunnoksista tietokoneen ja tallennuslaitteen välillä. Tiedostotason tallennusjärjestelmät hallitsevat itse omien massamuistiensa osoitusta, ja tarjoavat käytettyyn tiedostojärjestelmään tallennettua metadataa (tiedoston ja hakemiston nimi) tiedonsiirtoprotokollien yli tietokoneelle. Tietokone voi tämän metadatan avulla noutaa tiedostojen sisältämän datan NAS – laitteelta. (Meier 2008, Ruest et al. 2009)

Virtualisointi on muuttanut myös tietoliikennelaitteiden toimintaa. VLAN – tekniikka (IEEE 802.1Q) toi mukanaan mahdollisuuden määritellä 4094 virtuaalista verkkoa, joiden tunniste VID koostuu kahdestatoista bitistä ethernet – kehyksessä. Muutoksen ansiosta yhdessä verkkokaapelissa saattaa kulkea useaan eri layer 2:n verkkoon kuuluvaa dataa. (Farkas et al. 2013)

Monet käyttöjärjestelmät ovat myös ottaneet perinteisesti verkon aktiivilaitteille kuuluvia rooleja. Esimerkiksi Linux – käyttöjärjestelmä sisältää runsaasti työkaluja, joilla sitä voidaan käyttää OSI – mallin kerroksilla 2-4 (siirtokerros, verkkokerros ja kuljetuskerros) toimivana aktiivilaitteena. Toteuttaa voidaan esimerkiksi silta, kytkin, reititin tai L2/L3:n palomuuri. Vuonna 2009 julkaistiin Open vSwitch – ohjelmistopaketti, joka on tarkoitettu erityisesti virtualisointialustojen verkkovirtualisointiin. (Meier 2008, Pfaff et al. 2015)

2.4 Tärkeimmät virtualisointiin liittyvät luokittelut

Virtualisointia tapahtuu hyvin monella eri menetelmällä ja hyvin monenlaisissa järjestelmissä. Palvelinvirtualisoinnissa käytetyt menetelmät voidaan jakaa karkeasti kolmeen eri pääkategoriaan, jotka avataan tarkemmin seuraavissa aliluvuissa.

Virtualisointiytimet on perinteisesti myös jaoteltu kahteen eri pääluokkaan, joita käsitellään lyhyesti viimeisessä aliluvussa.

2.4.1 Täysvirtualisointi

Täysvirtualisoinnissa virtualisointiohjelmisto tarjoaa virtuaalikoneelle emuloidun laiteympäristön, jonka päällä vieraskäyttöjärjestelmää voidaan ajaa. Tyypillisesti emuloitavat laitteet on valittu niin, että niiden käyttöjärjestelmätuki on mahdollisimman laaja. Emulointi eristää vieraskoneelle näkyvän laitteiston todellisesta fyysisestä laitteistosta, ja näin ollen vieraskoneen käyttöjärjestelmä onkin täysin tietämätön tapahtuvasta virtualisoinnista. (Soriga et al. 2013)

Täysvirtualisoinnin perinteinen toteutustapa on ollut x86-arkkitehtuurin rajoituksista johtuen täysin ohjelmistopohjainen. Virtualisointiydin on tällöin joutunut huolehtimaan laitteiston emuloinnista, virtuaalikoneiden skeduloinnista, muistinhallinnasta ja muista resurssivarauksista. Yleensä ohjelmistopohjaiset virtualisointiohjelmistot ovat toimineet yleisten käyttöjärjestelmien, kuten Windows tai Linux, päällä.

Ohjelmistopohjaisen virtualisoinnin yleisenä ongelmana on sen hitaus. Esimerkiksi VMWaren aikoinaan kehittämä BT-tekniikka (Binary Translation) tarkkailee virtuaalikoneen suorittamia käskykannan käskyjä ja tarvittaessa korvaa laitteistoa liian rajoittamattomasti käsittelevät käskyt sarjalla uusia käskyjä, jotka eivät aiheuta ongelmia muiden virtuaalikoneiden tai itse virtualisointialustan kanssa. Tämä analyysi tapahtuu ajonaikaisesti, joten se aiheuttaa virtualisointialustalle ylimääräistä CPU - kuormaa sekä virtuaalikoneiden toimintaan viivettä. (VMWare 2009)

Puhtaasti ohjelmistopohjaista täysvirtualisointia käytetäänkin nykyään erittäin harvoin. Tähän on vaikuttanut erityisesti x86 - arkkitehtuurissa tapahtunut kehitys, jonka myötä prosessorien käskykantoja on laajennettu uusilla virtualisointia tukevilla tekniikoilla (Red Hat 2015). Ohjelmistopohjaisilla ratkaisuilla on kuitenkin etuna helppo siirrettävyys eri arkkitehtuurien välillä.

2.4.2 Paravirtualisointi

Paravirtualisoinnissa (engl. para-virtualized machine, PVM) lähestymistapa virtualisointiin on merkittävästi erilainen täysvirtualisointiin verrattuna. Siinä virtualisointiydin ei emuloi virtuaalikoneelle rautatason laitteita ja niiden rajapintoja, vaan tarjoaa oman rajapintansa virtuaalikoneiden käytettäväksi. Sen sijaan että virtuaalikone suorittaisi sarjan I/O – kutsuja emuloidulle laitteistolle, pyytää se virtualisointirajapinnan tarjoamien metodien avulla virtualisointiydintä suorittamaan tarvittavan operaation. Välistä jää siis laitteiston emulointiin käytettävä laskenta-aika ja sen aiheuttama kompleksisuus. Näin saman asian tekemiseen käytetään vähemmän käskyjä, joka puolestaan tarkoittaa vähemmän käytettyjä kellojaksoja ja viivettä virtualisointialustan laitteiston osalta. (Soriga et al. 2013)

Paravirtualisointi onkin merkittävästi ohjelmistopohjaista täysvirtualisointia nopeampaa ja tarjoaakin lähes natiiviin verrattavaa suorituskykyä virtualisointialustalla. Menetelmällä on kuitenkin hintansa; yleiset käyttäjärjestelmäytimet on suunniteltu käyt-

tämään laitteistoa suoraan, eikä virtualisointiytimen tarjoamaa rajapintaa. Näin ollen paravirtualisoinnissa joudutaan käyttämään sitä varten käännettyjä käyttöjärjestelmäytimiä. Tämä on ongelma erityisesti suljetun lähdekoodin käyttöjärjestelmien kanssa, sillä niiden toiminta paravirtualisoidussa ympäristössä vaatisi valmistajaa julkaisemaan tätä varten erikoisversion ytimestään. Näin ei ole tapahtunut. Sen sijaan avoimeen lähdekoodiin perustuvien käyttöjärjestelmien kanssa paravirtualisointi onkin ollut yleisempää. (Soriga et al. 2013, Xen 2016)

Käytännössä pelkästään paravirtualisointiin perustuvat virtualisointiohjelmistot ovatkin jääneet x86 - käskykannan virtualisointilaajennusten myötä pois käytöstä. Nykyaikaisissa virtualisointialustoissa paravirtualisointia käytetään kuitenkin laajasti osamenetelmänä erityisesti nopeiden I/O – laitteiden, kuten massamuistien ja verkkolaitteiden, virtualisoinnissa. Näillä paravirtualisoiduilla laitteilla ja laiteajureilla saavutetaan merkittävästi parempi suorituskyky emuloituihin laitteisiin verrattuna. Tyypillinen esimerkki on Red Hatin ja IBM:n kehittämä VirtIO – kehysrakenne, joka tarjoaa paravirtualisoidut laiteajurit verkkokorteille ja massamuisteille sekä Linux- että Windows-ympäristöön. (Red Hat 2015)

2.4.3 Laitteistoavusteinen virtualisointi

Laitteistoavusteinen virtualisointi (engl. hardware virtual machine, HVM) on ensimmäisen kerran vuonna 2005 esiteltyjen x86 - arkkitehtuurin virtualisointilaajennusten jälkeen muodostunut ylivoimaisesti käytetyimmäksi virtualisointimenetelmäksi (Red Hat 2015). Se perustuu ohjelmistopohjaisen täysvirtualisoinnin tavoin virtualisointiytimen tekemään laite-emulointiin, mutta prosessorien virtualisointilaajennusten hyödyntäminen siirtää suuren osan emulaatiota ohjelmistotasolta raudan hoidettavaksi. Näin saavutetaan paravirtualisoinnin tavoin lähes natiivia suorituskykyä vastaava nopeus. Laitteistoavusteinen virtualisointi ei emuloinnin takia kuitenkaan vaadi minkäänlaisia muutoksia virtuaalikoneessa ajettavaan käyttäjärjestelmäytimeen. Yleensä HVM - pohjaiset ratkaisut kykenevätkin ajamaan lähes kaikkia x86-yhteensopivissa tietokoneissa käytettyjä käyttöjärjestelmiä. (Soriga et al. 2013)

Yleisemmin käytössä olevat virtualisointiratkaisut käyttävät virtuaalikoneen perustoiminnan, kuten esimerkiksi prosessorin ja keskusmuistin emuloinnin suorittamiseen, laitteistoavusteista virtualisointia, ja näitä laitteita koskevat käskykannan käskyt suoritetaan suoraan oikealla alustan laitteistolla. Näiden tarjoama virtualisointituki takaa sen, etteivät eri käyttöjärjestelmäinstanssit vaikuta tahattomasti toistensa toimintaan. Oheislaitteita varten on yleensä valittavana joko ohjelmistopohjaisesti emuloidut tai paravirtualisoidut laitteet ja laiteajurit. Emuloitujen laitteiden etuna on hyvä yhteensopivuus, luotettavuus ja käytännössä koeteltu toiminta. Paravirtualisoitujen laitteiden etuna pidetään yleensä nopeampaa suorituskykyä, joskin esimerkiksi VirtIO – paketin laiteajurit ovat nykyään myös muun muassa Microsoftin WHQL – sertifikaatin saaneina todella luotettavia. (Soriga et al. 2013, Red Hat 2015)

2.4.4 Käyttöjärjestelmätasolla tapahtuva virtualisointi

Käyttöjärjestelmätasolla tapahtuva virtualisointi on ohjelmistopohjainen virtualisointimenetelmä, jossa palvelimen ajaman käyttöjärjestelmäytimen tarjoamat palvelut jaetaan erillisiin nimiavaruuksiin, jotka erottavat eri virtualisointi-instanssit toisistaan. Näin jokainen instanssi voi näyttäytyä user spacen ohjelmistoille omana palvelimenaan. Tavoitteena on, että nämä user spacessa ajettavat ohjelmistot eivät pysty kommunikoimaan suoraan toistan instanssien ohjelmistojen kanssa, vaan kaikki kommunikaatio tapahtuu virtualisointirajapinnan kautta. Tyypillisiä käyttöjärjestelmätason virtualisointia hyödyntäviä ohjelmistoja ovat muun muassa FreeBSD – käyttöjärjestelmässä käytettävät Jail – tekniikka, Chroot, LXC (Linux containers) ja Docker. Kuvatuntyyppistä virtualsointiinstanssia kutsutaankin yleensä nimellä kontaineri (engl. container). (Ruest et al. 2009, Reshetova et al. 2014, Grattafiori 2016)

Käyttöjärjestelmätason virtualisointi koostuu hyvin ohuesta ohjelmistorajapinnasta ajettavan instanssin ja alustana toimivan käyttöjärjestelmäytimen välissä. Laitteiston emulointia ei tarvita, sillä rajapinnalle tehdyt käyttöjärjestelmätason kutsut ajetaan alustana toimivan käyttöjärjestelmän kautta. Tämä tekee käyttöjärjestelmätason virtualisoinnista hyvin nopeaa ja tehokasta. Tämä käytäntö tarkoittaa samalla sitä, että virtualisointi-instanssien käyttöjärjestelmätuki on rajattu samaan versioon, joka alustakoneella on käytössä. (Reshetova et al. 2014, Grattafiori 2016)

Menetelmän nopeuden tarjoava etu, jaettu alustakoneen käyttöjärjestelmäydin, on samalla myös merkittävä tietoturvariski: verrattuna täysvirtualisointiin ja paravirtualisointiin, käyttöjärjestelmätason virtualisointi paljastaa virtuaali-instansseille erittäin suuren hyökkäyspinnan alustakoneen omiin palveluihin. Jaetun ytimen myötä myös siinä olevat tietoturvaongelmat ovat yhteisiä virtualisointialustalle ja sen virtuaali-instansseille. Iso osa käyttöjärjestelmätason virtualisoinnin turvaongelmista aiheutuukin juuri jaetun ytimen järjestelmäkutsuista sekä informaatiovuodoista ja jaetusta laiterajapinnasta. (Reshetova et al. 2014)

Nämä tietoturvariskit muuttuvat kuitenkin eduksi, kun niitä vertaillaan muiden virtualisointimenetelmien sijaan normaaliin käyttöjärjestelmän päällä toimivaan ohjelmistoon: potentiaalinen hyökkäysala on tällöin pienempi, sillä käyttöjärjestelmätason virtualisointin käyttämä rajapinta rajoittaa ohjelmistojen toimintaa enemmän kuin pelkkä käyttöjärjestelmäydin. Tätä tapaa hyödynnetään muun muassa Linuxiin perustuvassa Googlen Android – käyttöjärjestelmässä parantamaan mobiililaitteiden ja niiden sovellusohjelmiston tietoturvaa. (Reshetova et al. 2014)

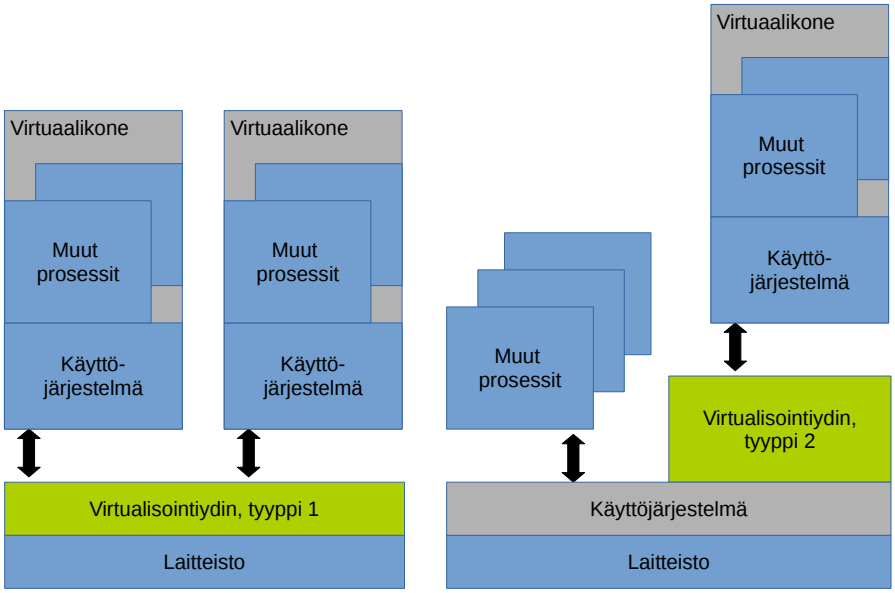
Käyttöjärjestelmätason virtualisointi on Docker – ohjelmistossa oivallettu myös eduksi, jonka avulla voidaan helpottaa ohjelmien siirrettävyyttä tietokoneesta ja käyttöjärjestelmästä toiseen. Docker toteuttaa tämän keräämällä ohjelmiston ja kaikki sen tarvitsemat komponentit yhteen helposti siirrettävään omavaraiseen pakettiin. Tätä ohjelmistopakettia voidaan ajaa niin sovelluskehittäjän kannettavalla tietokoneella, yrityksen

omilla palvelimilla, pilvipalvelun tarjoaman Docker – palvelun päällä tai suoraan sulautetun järjestelmän päällä. (Noyes 2013)

2.4.5 Virtualisointiytimen luokittelu

Perinteisesti virtualisointiytimet on luokiteltu kahteen eri päätyyppiin 1 ja 2. Gerald Popek ja Roberd Goldberg kuvailivat näitä päätyyppejä vuonna 1974 julkaistussa, hyvin matemaattisessa artikkelissaan ”Formal Requirements for Virtualizable Third Generation Architechtures”. Termit tyyppi 1 ja tyyppi 2 yleistyvät käyttöön kuitenkin vasta myöhemmin. (Liguori 2007)

Kuvassa 2.1 on esitelty nämä arkkitehtuurityypit tarkemmin:



Kuva 2.1: Kuvassa virtualisointiytimen tyypit 1 ja 2. Mukaillen (Milberg et al. 2013)

Määritelmän mukaan tyypin 1 virtualisointiydin toimii suoraan laitteiston alustan päällä ilman käyttöjärjestelmää ja hallitsee laiteresursseja itse. Tästä johtuen tyypin 1 virtualisointiytimiä kutsutaan myös nimellä natiivi virtualisointiydin (engl. native hypervisor). Virtuaalikoneet toimivat virtualisointiytimen päällä prosesseina. (Milberg et al. 2013)

Tyypin 2 virtualisointiydin puolestaan vaatii määritelmän mukaan alleen käyttöjärjestelmän hoitamaan alustakoneen laiteresursseita keskittyen itse virtuaalikoneiden hallinnointiin. (Liguori 2007)

Todellisuudessa nämä erot eivät ole kuitenkaan kovin selviä. Jokainen tyypin 1 virtualisointiohjelmisto nimittäin vaatii alleen myös jonkinasteisen käyttöjärjestelmän ja

ytimen ohjaamaan laitteiston ja ohjelmistokomponenttien toimintaa. Käytännössä tyypillä 1 viitataankin omaan hyvin pelkistettyyn mikroytimeensä turvautuvaan ohjelmistoon. Tyypillistä tämänkaltaisille ohjelmistoille on lisäksi se, että ne tarvitsevat toimintaansa varten myös normaalin käyttöjärjestelmän yhdeksi hallintavirtuaalikoneeksi. Esimerkiksi Xen vaatii hallintakoneeksi, josta käytetään Xen - terminologiassa nimitystä dom0, Linuxiin, Solarikseen tai BSD:hen pohjautuvan käyttöjärjestelmän. Dom0 käynnistyy automaattisesti virtualisointialustan käynnistyessä. Sitä ajetaan virtuaaliprosessoreilla Xenin mikroytimen päällä, mutta muista virtuaalikoneista poiketen sillä on rajoittamaton pääsy alle olevaan laitteistoon. Tätä pääsyä tarvitaan, sillä dom0:n vastuulla on tarjota laiteajurit alustan I/O – laitteille. Tämä tehdään hallintakoneessa, jotta virtualisointiin kuulumattomat asiat eivät vaikuttaisi varsinaisen virtualisointiytimen suorituskykyyn. (Liguori 2007, Soriga et al. 2013)

Puhtaimmillaan tyypin 2 virtualisointiratkaisuja edustaa ohjelmistopohjaista täysvirtualisointia hyödyntävät ohjelmistot, joita käytettiin x86 - virtualisoinnin alkutaipaleilla. Näitä ovat muun muassa VMWaren Workstation ja ESX Server – tuoteperheissä käytetyt VMKernelin varhaiset versiot sekä avoimen lähdekoodin Qemu. Nykyään VMWaren tuotteissa käytetään laitteistopohjaiseen virtualisointiin perustuvaa menetelmää. Qemu - projekti puolestaan on laitteistoemuloinnin toteuttava osa KVM - virtualisointiohjelmistossa. (Liguori 2007, VMWare 2007, VMWare 2009)

Virtualisointiytimien tyyppiluokittelutkin ovat nykyään menettäneet suurelta osin merkityksensä, sillä laitteistopohjaisen virtualisoinnin myötä puhtaasti tyyppien 1 ja 2 mukaiset virtualisointiratkaisut käytännössä hävisivät. Nykyään käytettävät ratkaisut voidaan määritelmien mukaan sijoittaa kumpaan tahansa tyyppiluokkaan. (Liguori 2007, Soriga et al. 2013)

2.5 Virtualisoinnit edut

Virtualisointi tuo mukanaan monia etuja, joista merkittävimpiä on laitteistojen käyttöasteen nousu. Normaalisti palvelimien käyttöasteet ovat noin 5% - 15% luokkaa, mutta virtualisoiduissa ympäristöissä fyysisen laitteiston käytössä päästään jopa 70% käyttöasteisiin (Meier 2008). Tietyissä alustoissa käyttöasteeksi suositellaan jopa 80% kuormaa (IBM 2012). Tällä on merkittäviä vaikutuksia laitteistokuluihin, mutta suurimmat säästöt yleensä syntyvät vähentyneestä sähkönkulutuksesta ja siten myös vähentyneestä jäähdytyksen tarpeesta. (Ruest et al. 2009)

Virtualisoitaessa alustakoneen prosessorimäärä voidaan myös ylivarata. Jos keskimääräinen kuormitus on luokkaa 5% - 15%, voidaan yhdellä palvelimen prosessorilla teoriassa suorittaa kymmenen virtuaaliprosessorin ajoa. Tällöin kuormitus on 50% - 150% prosessorin suorituskyvystä, eli normaalijakauman mukaan todennäköisesti noin 100% luokkaa. Yksittäisen virtuaaliprosessorin kuormaa on kuitenkin todella vaikea ennustaa, joten Red Hatin best practice – suosituksen mukaan tätä kymmenen virtuaalipro-

sessorin ja yhden todellisen prosessorin ylivarauksen suhdetta ei tulisikaan ylittää. (Herrmann et al. 2016)

Fyysisen tietokonelaitteiston abstrahointi puolestaan mahdollistaa virtuaalipalvelimien laitevakioinnin ja niiden laitteistokannan yksinkertaistamisen. Kun virtuaalikoneille emuloidaan vain tarkasti harkittu laitevalikoima, tai käytetään paravirtualisoituja laiteajureita, vähennetään samalla virtuaalikoneilla tarvittavaa ylläpidon tarvetta ja laiteajurien määrää. Tämä yksinkertaistaa virtuaalikoneilla tehtävää ylläpitotyötä ja helpottaa mahdollisissa ongelmatilanteissa vianselvitystä. (Meier 2008)

Virtualisointijärjestelmien monikerroksinen rakenne vähentää myös osaltaan virtuaalikoneiden laitekompleksisuutta. Jos virtualisointialustalla on käytössä vikasietoinen levyjärjestelmä, virtuaalikoneeseen yhdistetyille levyille on yleensä täysin tarpeetonta luoda minkääntasoista vikasietoisuutta, vaan yksinkertaisemman ohjaimen emulointi (IDE, SCSI) tai esimerkiksi paravirtualisoidun levyohjaimen käyttö riittää. Virtuaalikonetasolle luotu vikasietoisuus yleensä vain aiheuttaakin alustalle enemmän laskettavaa ja täten hidastaa toimintaa ilman että datan turvallisuus millään tapaa lisääntyisi. Useimmissa virtualisointijärjestelmissä virtuaalikoneelle voidaan läpivientinä antaa kokonainen PCIE - laite, esimerkiksi RAID - ohjain ja siihen kytketyt levyt. Tällöin virtuaalikoneen massamuistiohjaimen suorituskyky on hyvin lähellä pelkän rautapohjaisen vaihtoehdon suorituskykyä, luokkaa 96% - 99%. Koneen siirto toiselle alustalle on tällöin kuitenkin merkittävästi vaikeampaa, sillä PCIE -läpivienti on kytketty laitteen PCI - laitetunnukseen. (Walters et al. 2014)

Laitekannan abstrahointi vaikuttaa virtuaalikoneiden toimintaan edullisesti hyvin monella tasolla. Kuvassa 2.2 on esitelty kolmesta virtualisointialustasta koostuva hyvin heterogeeninen klusteri ja neljä sen päällä toimivaa virtuaalikonetta. Virtualisointijärjestelmänä toimii tässä tapauksessa KVM, joskaan se ei ole esimerkin kannalta merkityksellistä.

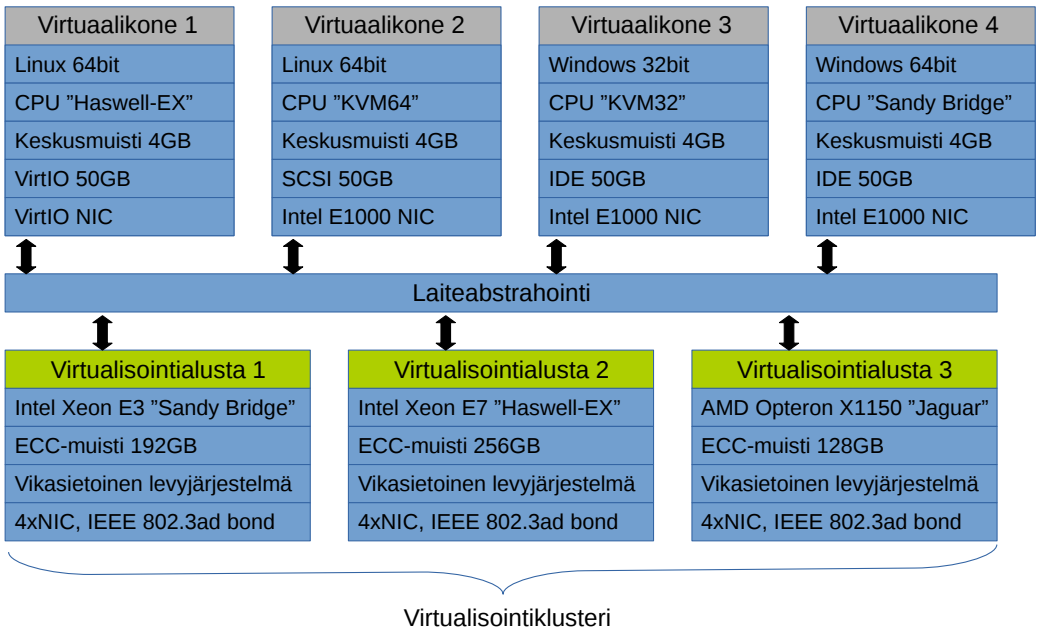
Tarkastellaan ensimmäisenä verkkoabstraktion vaikutusta järjestelmään. Virtuaalikoneissa 2 – 4 verkkokorttina on emuloitu Intel E1000 – piiriin perustuva ohjain. Kyseinen verkkolaite on hyvin yleinen, ja sille löytyykin suora ajurituki lähes kaikista yleisimmistä käyttöjärjestelmistä. Näin ollen virtuaalikoneen ylläpitäjän ei tarvitse erikseen asentaa tai ylläpitää laiteajuria.

Virtuaalikoneessa 1 käytetään paravirtualisoitua VirtIO – paketin verkkoajuria, jolle löytyy laiteajuri suoraan Linuxin ytimestä. Laiteajuri on saatavilla myös monelle muulle käyttöjärjestelmälle, mutta se pitää erikseen asentaa niihin.

Virtualisointialustoilla on puolestaan neljä verkkolaitetta, jotka on yhdistetty IEEE 802.3ad – standardin mukaisesti yhdeksi vikasietoiseksi ja kuormaa tasaavaksi verkkolaitteeksi, jota tässä tapauksessa kutsutaan nimellä bond0. Virtualisointiydin huolehtii siitä, että virtualisoitujen verkkolaitteiden liikenne kulkee alustojen laitteen bond0 kautta. Virtuaalikoneen 1 ja muiden virtuaalikoneiden erona on suorityskyky; oikeiden verk-

kokorttien emulointi vaatii enemmän CPU – aikaa alustalta ja aiheuttaa verkkoliikenteeseen enemmän viivettä kuin paravirtualisoidun laiteajurin käyttäminen.

Virtualisointialustojen fyysisten verkkolaitteiden tyyppiä ei ole kuvaan merkitty, eikä sillä ole virtuaalikoneiden kannalta mitään merkitystä. Abstrahoinnin etuna verkkoliikenteen tapauksessa onkin se, että virtuaalikoneiden ei tarvitse tietää mitään alla olevasta laite- tai softapinosta, vaan niille riittää esimerkin tapauksessa kahden vakioidun verkkosovittimen liikenteen hallinta.



Kuva 2.2: Virtualisoinnin edut, laiteabstrahointi

Siirrytään tarkastelemaan kuvan laitteistojen prosessorivalikoimaa. Alustoissa on käytössä hyvin erityyppiset prosessorit, eikä kuvattu tilanne olekaan todellisuudessa suositeltu vaihtoehto. Itse asiassa, alustojen erityyppisistä prosessoreista johtuen kaikkia esimerkin virtuaalikoneita ei edes voida ajaa niissä määritetyillä prosessorityypeillä, sillä virtualisointialusta 3:n AMD Opteron ei sisällä kaikkia niitä ominaisuuksia, joita virtuaaliprosessoreille on määriteltynä. Prosessorityypeistä löytyy lisätietoa liitteestä A, jossa on listattu myös kaikki KVM:n tukemat prosessorien ominaisuudet (engl. CPU flags). Eri prosessorisukupolvet toteuttavat aina tietyn osajoukon kaikista listatuista ominaisuuksista. (Gomez-Folgar et al. 2014)

Alla on taulukoitu mitkä virtuaalikoneet toimivat milläkin virtualisointiklusterin eri alustakoneella:

Sopivat virtuaalikoneet
Virtualisointialusta 1: koneet 2, 3 ja 4
Virtualisointialusta 2: koneet 1, 2, 3, ja 4
Virtualisointialusta 3: koneet 2 ja 3

Käytännössä asia ei ole aivan näin joustamaton. Useimmat virtualisointijärjestelmät joko varoittavat yhteensopimattomista alustaprosessoreista ja/tai ohjaavat käyttäjän automaattisesti käyttämään virtuaalikoneissa suurimman mahdollisen yhteisen osajoukon toteuttavaa prosessorityyppiä. Esimerkin tapauksessa tämä olisi KVM64 - tyypin prosessori. Näin CPU - abstrahointi mahdollistaa hyvinkin erityyppisten prosessorien käytön alustana toimivissa koneissa. Virtuaalikoneiden suorituskykyyn tällä saattaa olla negatiivinen vaikutus, sillä tällöin osa tuoreamman prosessorin uusista ominaisuuksista jää hyödyntämättä. Käytännössä ero kuitenkin näkyy vain hyvin erikoistuneissa ja CPU – intensiivisissä ohjelmistoissa.

Keskusmuistin ja massamuistien kohdalla abstraktiotason etu on hyvin samantyyppinen. Virtuaalikoneiden ei ole hyödyllistä toteuttaa vikasietoisuutta, jos alustakoneilla käytetään jo vikasietoisia järjestelmiä. Massamuistitilaa voidaan virtualisointijärjestelmissä lähes poikkeuksetta osoittaa hyvin erityyppisistä laitteista. Näitä vaihtoehtoja käsitellään myöhemmin tarkemmin.

2.6 Virtualisoinnin haasteet, haitat ja potentiaaliset ongelmakohdat

Virtualisointi lisää aina tietokonejärjestelmään uusia abstraktiotasoja ja näiden välistä ohjausliikennettä. Virtualisoinnin avulla ei voida milloinkaan tavoittaa samaa 100% suoritustasoa verrattuna laitteiston päällä pyörivään yksittäiseen ohjelmaprosessiin. Mikäli ajettavia prosesseja on kuitenkin monta, esimerkiksi useampia käyttöjärjestelmiä, niiden yhteenlaskettu kyky hyödyntää alla olevaa laitteistoa voi kuitenkin olla yksittäistä prosessia huomattavasti korkeampi, sillä laitteisto voi rinnakkaisesti suorittaa useampia tehtäviä esimerkiksi hitaan massamuistin I/O - liikenteen aikana. (Wang et al. 2010)

Näin ollen suurta laskentakapasiteettia kaipaavaa palvelua ei lähtökohtaisesti ole tehokasta ajaa virtualisointialustan päällä, vaan se tulisikin ajaa aina omalla dedikoidulla raudalla. Samaa pätee palveluihin, jotka edellyttävät valtaisaa I/O - liikennettä paikallisille massamuisteille. Uudet tekniikat, kuten prosessoriarkkitehtuurien virtualisointilaajennukset, virtualisointijärjestelmien PCIE - läpiviennit ja entistä nopeammat SAN- ja NAS- tallennusjärjestelmät ovat kuitenkin parantaneet mahdollisuuksia virtualisoida myös edellä mainittuja raskaampiakin palveluita. (Meier 2008)

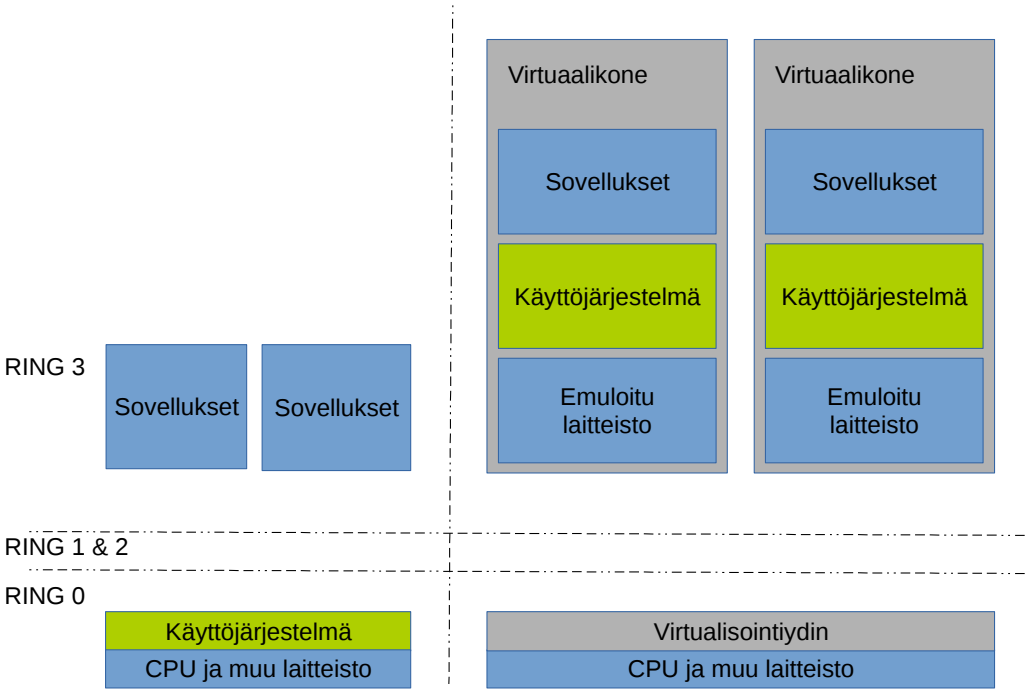
Teknisten, suorituskykyä parantavien tekniikoiden kanssa on silti syytä olla myös varovainen. Nykyaikaiset levyjärjestelmät esimerkiksi mahdollistavat tallennustilan ylivaraamisen käytettävissä olevaan kokonaiskapasiteettiin nähden. Kun virtuaalikoneiden tallennustilan tarve kasvaa, voi levyjärjestelmän levykapasiteetti loppua yllättäen kes-

ken. Tämä voi johtaa virtuaalikoneiden kaatumiseen tai jopa niiden datan korruptoitumiseen. Samaa ylivaraamista voidaan tehdä myös keskusmuistin tai prosessoriytimien suhteen. Virtuaalikoneille voi yhteensä olla helposti määriteltynä merkittävästi enemmän virtuaaliytimiä kuin alustana toimivassa raudassa prosessoriytimiä on tarjolla. Tämän vaikutukset kuitenkin yleensä rajoittuvat poikkeustilanteissa esiintyviin suorituskykyongelmiin. (Milberg et al. 2013)

2.7 CPU:n Ring-levelit ja x86-tekniikan erityispiirteet

Virtualisointi on muuttanut tapaa, jolla organisaatiot rakentavat ja hallitsevat IT – infrastuktuuriaan. X86 - pohjainen prosessoriarkkitehtuuri on yleistynyt sille tasolle, että se käytännössä hallitsee markkinoita. Arkkitehtuurin rajoitteet olivat kuitenkin pitkään este virtualisoinnille.

Merkittävin näistä rajoitteista oli x86 – arkkitehtuurin suojaustasot, jotka tunnetaan paremmin nimellä *ring levels*. Niiden tarkoituksena on tarjota ohjelmistoille turvallinen käyttöympäristö, jossa käyttäjien sovellukset erotetaan käyttöjärjestelmästä. Tässä menetelmässä prosessori tarjoaa neljä eri suojaustasoa (ring 0 – ring 3). Alimman tason ring 0 on etuoikeutetuin prosessorin suojaustaso, joka tarjoaa täyden pääsyn prosessoriin ja muuhun laitteistoon. Käytännössä tämä tarkoittaa sitä, että ring 0 - suojaustasolla voidaan käyttää kaikkia x86 – prosessoriarkkitehtuurin käskyjä. Perinteisesti tämän tason toiminta on varattu käyttöjärjestelmälle. Suojaustasot 1 – 3 olivat puolestaan toinen toistaan rajoitetumpia. Yleisimmissä käyttöjärjestelmissä, kuten Linux ja Microsoft Windows, käyttöjärjestelmäydin toimii suojaustasolla 0 ja sovellukset tasolla 3. Suojaustasot 1 ja 2 eivät ole käytössä. (Duarte 2008, Red Hat 2015)



Kuva 2.3: Kuvassa x86 - tekniikan ring levelit. Vasemmalla perinteinen malli, oikealla täysvirtualisoinnissa käytetty malli. Mukaillen (Red Hat 2015)

Tämä suojausmalli on hyödyllinen, kun laitteistolla ajetaan vain yhtä käyttöjärjestelmää. Virtualisoinnissa kuitenkin virtualisointiytimen pitää toimia suojaustasolla 0, sillä se hallitsee käyttöjärjestelmien sijaan laitteistoa ja järjestelmäkutsuja. Täysvirtualisoinnissa virtuaalikoneet sijaitsevat, kuten kuvasta 2.3 nähdään, suojaustasolla 3. Tämä aiheuttaa ongelman: käyttöjärjestelmät on alun perin suunniteltu toimimaan suojaustasolla 0. Mikäli virtuaalikone yrittäisi toimia prosessorin suojaustasolla 0, johtaisi se todennäköisesti virtuaalikoneen kaatumiseen. (Duarte 2008, Red Hat 2015)

Ohjelmistopohjaisessa täysvirtualisoinnissa ongelma on ratkaistu tyypillisesti VMWaren kehittämän Binary Translation – tekniikan tavoin: virtualisointiydin tarkkailee virtuaalikoneiden muistialueita, ja mikäli suojaustason 0 käskyjä havaitaan, virtualisointiydin korvaa ajettavan laitekoodin sarjalla uusia käskyjä niin, että suojaustason 0 käskyjä ei tarvita. (Red Hat 2015)

Paravirtualisoinnissa on käytetty erityyppistä lähestymistapaa ongelmaan. Siinä virtuaalikoneessa käytetään paravirtualisointia varten muokattua käyttöjärjestelmäydintä, joka siirtää suojaustason 0 käskyt virtualisointiytimen suoritettavaksi. (VMWare 2009, Red Hat 2015)

Nämä ratkaisut jäivät kuitenkin käytännössä historiaan. Prosessorivalmistajat ymmärsivät virtualisointikyvyn tärkeyden, ja laajensivat x86-käskykantaa uusilla virtualisointia helpottavilla laajennuksilla luoden samalla mahdollisuuden laitteistoavusteiselle

virtualisoinnille. Intel toi markkinoille VT-X – laajennuksen vuonna 2005 ja AMD puolestaan oman AMD-V – laajennuksensa vuonna 2006. Laajennukset eivät ole keskenään yhteensopivia, mutta niiden perustoimintaperiaatteet ovat hyvin samankaltaiset. Ne tarjoavat seuraavanlaisia ominaisuuksia:

i) Molemmat laajennukset toivat prosessorille uuden toimintatilan, vierastilan. Vierastila voi normaalitilan tapaan toimia millä tahansa prosessorin oikeustasolla ilman rajoituksia. Erona normaalitilaan on kuitenkin se, että virtualisointiydin voi antaa prosessorille listan käskykannan käskyistä ja rekistereistä, joiden käyttöpyrkimykset vierastilassa palauttavat kontrollin virtualisointiytimelle. Näin virtualisointiydin voi välttää ristiriidat ajettavien instanssien välillä.

ii) Laajennukset tuovat mukanaan laitteistotilan vaihtomekanismin. Tämän avulla prosessorien rekisterien toimintatilaa voidaan vaihtaa tarvittavan tilan mukaan.

iii) Molemmissa laajennuksissa on myös määriteltynä mekanismi, joka raportoi virtualisointiytimelle aina kun vierastilasta poistutaan ja miksi sieltä on poistuttu. Näin virtualisointiydin voi käsitellä poistumissyyn ja suorittaa tarvittavat toimenpiteet. (Kivity et al. 2007, Red Hat 2015)

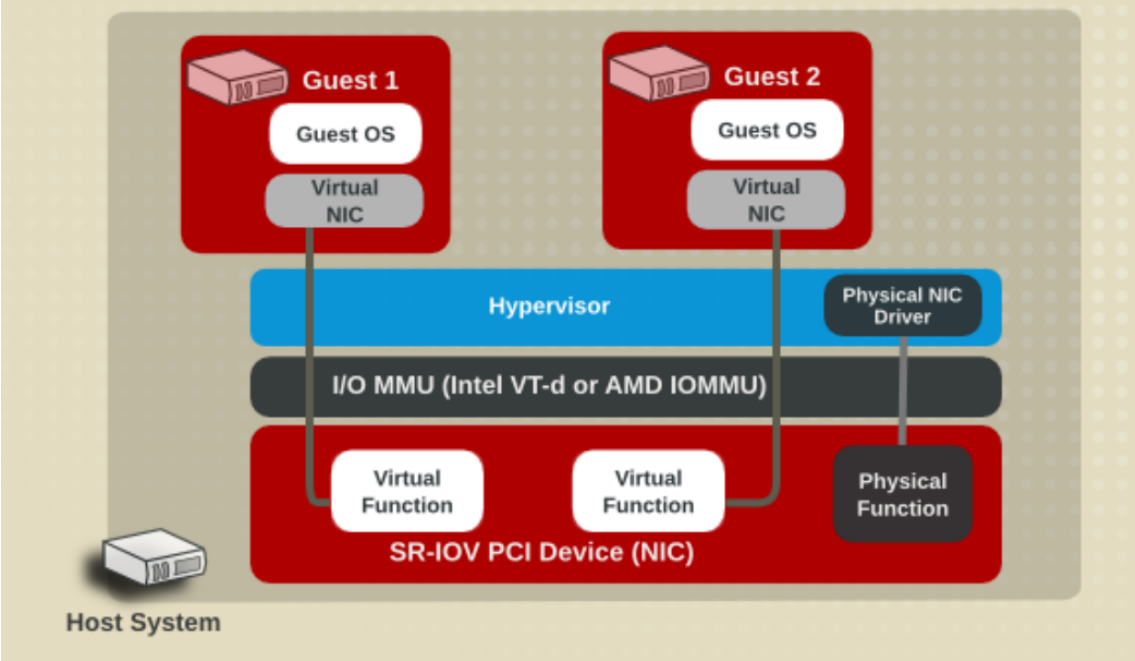
Virtualisointilaajennusten ansiosta prosessorin ei tarvinnut virtuaalikoneiden lisäksi ajaa Binary Translation – tekniikan tyyppistä ohjelmakoodia, mikä paransi suorituskykyä merkittävästi. Edelleen virtualisointialusta käytti kuitenkin paljon resursseja keskusmuistin virtualisoimiseen. Koska virtuaalikoneet eivät voineet suoraan käsitellä keskusmuistia, virtualisointiytimen pitää tarjota niiden käyttöön menetelmä, joka yhdistää virtuaalikoneen käyttämän muistiavaruuden osoitteet virtualisointialustan keskusmuistin todellisiin muistiosoitteisiin. Tämä on usein toteutettu virtualisointijärjestelmissä ohjelmallisesti käyttäen shadow page tables – tekniikkaa. (Red Hat 2015)

Ohjelmallinen ratkaisu on kuitenkin hidas, joten prosessorivalmistajat rakensivat prosessoreihinsa MMU-yksikön (engl. Virtualized memory management unit). AMD – kutsuu tekniikkaa RVI:ksi (Rapid Virtualization Indexing) ja Intel puolestaan EPT:ksi (Extended Page Table). Molemmat menetelmät tarjoavat merkittävää nopeusetua ohjelmalliseen ratkaisuun verrattuna. (Red Hat 2015)

Molemmat x86 – valmistajat ovat myös laajentaneet prosessoriensa toiminnallisuutta (AMD:n IOMMU ja Intelin VT-D) helpottamaan I/O – liikenteen virtualisoinnin käsittelyä. Tämä vaikutus näkyy erityisesti PCI – läpiviennin kohdalla, jossa virtualisointialustan PCI- tai PCIE – laite annetaan suoraan virtuaalikoneen hallintaan. Tämä tarjoaa natiivia suorituskykyä esimerkiksi RAID – ohjaimen läpiviennillä nopeaa levykapasiteettia tarvittaessa. (Red Hat 2015)

2.8 SR-IOV

SR-IOV – tekniikka (engl. Single Root I/O Virtualization) on PCI-SIG – ryhmittymän (PCI Special Interest Group) kehittämä standardi, jonka avulla yksittäinen PCIe – laite voidaan jakaa usean virtuaalikoneen käyttöön. Sen tavoitteena on parantaa PCIe – laitteiden suorituskykyä virtualisoitaessa. (Herrmann et al. 2016)



Kuva 2.4: SR-IOV - tekniikka mahdollistaa yhden PCIe - laitteen jaon useammalle virtuaalikoneelle. (Herrmann et al. 2016)

SR-IOV mahdollistaa sitä tukevan laitteen näkymisen useampina erillisinä fyysisinä laitteina virtualisointialustassa. Tätä varten SR-IOV määrittelee kaksi erityyppistä toimintatilaluokittelua laitteille:

i) Fyysiset funktiot PF (engl. Physical Functions) ovat rautatason toimintoja PCIe – laitteilla. Näillä viitataan normaaleihin käyttöjärjestelmälle tai virtualisointiytimelle näkyviin PCIe – laitteisiin ja niiden tarjoamiin palveluihin.

ii) Virtuaaliset funktiot VF (engl. Virtual Functions) toimivat rajapintana virtualisointikerroksen ja PCIe – laitteiden fyysisten funktioiden välillä.
(Herrmann et al. 2016)

Jokaisella virtuaalisella SR-IOV - laitteella on oma laitekonfiguraatio ja osoitteisto jotka virtuaaliset funktiot tarjoavat. Käytännössä ne säilyttävät virtualisoinnissa tarvittavia tietoja (kuten laiteosoitteita) ja käsittelevät virtualisoitua I/O – liikennettä niin, että jokainen VF on täysin eriytetty muista VF:stä ja PF:n toiminnasta. (Herrmann et al. 2016)

Jokainen VF voidaan jakaa suoraan virtuaalikoneiden käyttöön, esimerkiksi verkkokortin tapauksessa yksi PF voi tarjota useamman VF:n virtuaalikoneille käytettäväksi. VF näkyy virtuaalikoneessa normaalina verkkokorttina. Tällöin kyseisen VF:n laitekonfiguraatio ja -tila näkyy vain tälle virtuaalikoneelle. Tämä jakaminen edellyttää virtualisointialustalta VT-D tai IOMMU – tekniikan tukea. (Herrmann et al. 2016)

Linux-ydin sisältää tuen SR-IOV – tekniikalle. Tämä tuki on implementoitu ytimen PCI – alijärjestelmässä. Pelkkä ytimen tuki ei kuitenkaan riitä, vaan laiteajureiden on myös tuettava tekniikkaa. (Herrmann et al. 2016)

SR-IOV – tekniikan etuihin kuuluu lähes natiivi suorituskyky. Sen tarjoama laitteiden virtualisointituki on tärkeää etenkin palvelinkeskuksissa, jolloin niissä pystytään tarjoamaan laitepalveluita fyysisesti entistä pienempään tilaan mahtuvalla laitteistolla. SR-IOV – tekniikkaa tukevia laitteita ei kuitenkaan vielä ole laajalti markkinoilla. (Herrmann et al. 2016)

3 KERNEL-BASED VIRTUAL MACHINE

KVM, Kernel-based Virtual Machine, edustaa avoimen lähdekoodin uusimman sukupolven virtualisointityökaluja, joka on alusta asti tukeutunut rautatason virtualisointitukeen. KVM - projektin tavoitteena onkin ollut kehittää nykyaikainen virtualisointiydin (engl. hypervisor), joka luottaa vahvasti aiempaan hyväksi todettuun kehitystyöhön sekä laitteistokehityksen viimeisimpiin saavutuksiin. Projektin aloitti Avi Kivity Qumranet - yhtiössä. Kuitenkin jo vuonna 2008 Red Hat osti Qumranetin ja jatkoi vahvasti KVM:n kehitystyötä. Tämä panostus ja se, että KVM saavutti vuonna 2007 Linux-ytimen kehittäjien yleisen hyväksynnän, on vaikuttanut ohjelmiston asemaan merkittävästi. Se onkin ollut Linux - ytimen virallinen osa versiosta 2.6.20 – lähtien. Nykyään myös moni muu IT-alan suuri toimija, kuten IBM, HP, AMD, Intel, Novell, Siemens ja SGI panostaa KVM:n jatkokehitykseen. Se on myös tekniikkana mukana monessa pilvipalveluita tuottavissa ohjelmistoissa, mukaanlukien OpenNebula, Eucalyptus, Apache CloudStack, Google Compute Engine, IBM RC^2, IBM SmartCloud ja Red Hat OpenShift (Chen et al. 2011, Gomez-Folgar et al. 2014). (Burns 2010, Red Hat 2015)

3.1 Sovelluspinosta koostuva virtualisointiohjelmisto

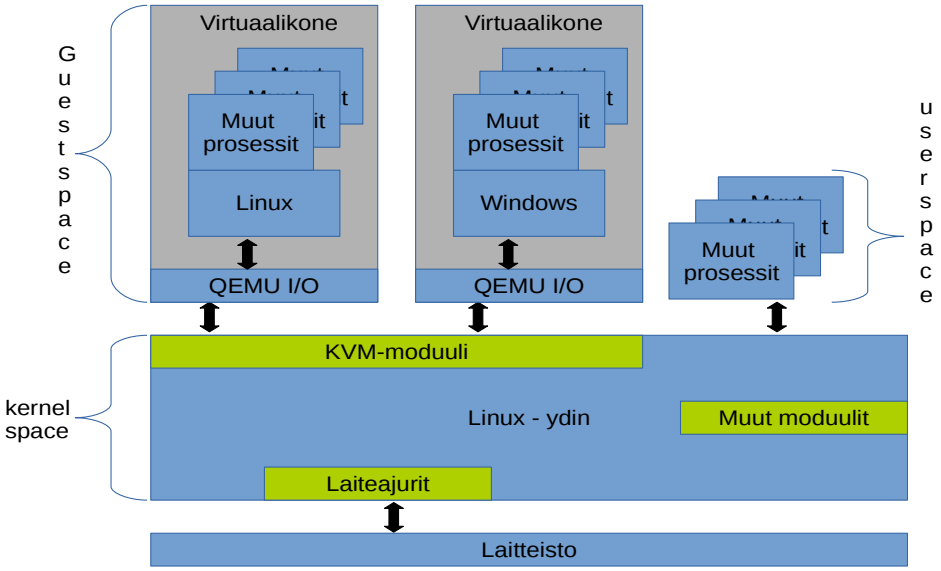
KVM - projektin kaksi tärkeintä suunnitteluperiaatetta ovat sen vakauden ja hyvän suorituskyvyn lisäksi auttaneet sitä nopeasti saavuttamaan ykkösaseman avoimen lähdekoodin virtualisointiohjelmistona. Ensinnäkin, koska KVM suunniteltiin vasta x86 – käskykannan virtualisointilaajennusten julkaisun jälkeen, sen ei ole missään vaiheessa tarvinnut toteuttaa nykyään rautatason tarjoamia palveluja, kuten virtuaalikoneiden matalan tason käskyjen kaappaamista ennen niiden suorittamista. Tämä vähentää merkittävästi menneisyyden aiheuttamaa painolastia, joka konkreettisesti näkyy virtaviivaisempana arkkitehtuurina ja pienempänä koodimääränä. VT-X ja AMD-V – laajennuksiin nojautumisen takia KVM:n ei ole missään vaiheessa tarvinnut vaatia ajettavien virtuaalikoneiden muokkaamista sille sopivaksi, vaan käyttöjärjestelmiä on voitu ajaa sellaisena kuin valmistaja on ne toimittanut. (Kivity et al. 2007, Red Hat 2015)

Toiseksi, KVM - projekti on rakentunut jo olemassa olevaan perustaan luottaen. Virtualisointiytimelle ei riitä ainoastaan kyky hyödyntää laitteistotason virtualisointitukea. Sen pitää muun muassa myös kyetä hallitsemaan alla toimivan laitteiston muistia ja I/O - pinoa, skeduloida ajamiaan prosesseja, tarjota laiteajurit alustan laitteille ja hallita verkkoprotokollapinoa. Itse asiassa, virtualisointiydin ja -ohjelmisto on aina poikkeuksetta erikoistunut käyttöjärjestelmä eroten yleiseen käyttöön tarkoitetuista käyttöjärjes-

telmistä lähinnä sen suhteen, että se ajaa normaalien prosessien lisäksi myös virtuaalikoneita ja tarjoaa niille emuloituja laiteresursseja. (Red Hat 2015)

Kaikki nämä KVM - projektin tarvitsemat resurssit löytyivät jo hyvin testattuina ja koeteltuina Linux-ytimestä sekä Qemu - projektista, ja nykyään KVM - moduuli onkin Linux - vakioytimen oleellinen osa.

Kuvassa 3.1 on esitetty tämä sovelluspino ja sen väliset suhteet:



Kuva 3.1: KVM-arkkitehtuuri. Linux-ydin ja KVM-moduuli muodostavat yhdessä virtualisointiytimen, joka tarjoaa palveluja ylemmille kerroksille.

Virtualisointiytimen muodostavat Linux ja sen KVM - moduuli tarjoavat palvelua niin normaaleille Linux-prosesseille kuin virtuaalikoneille. Itse asiassa jokaisen virtuaalikoneen jokainen virtuaaliprosessori näkyy järjestelmässä omana prosessinaan, ja niitä voi hallita kuten kaikkia muitakin prosesseja.

3.1.1 KVM, Linux-ytimen moduuli

Edellisessä aliluvussa mainittiin, että KVM -moduuli ja Linux - ydin muodostavat yhdessä virtualisointiytimen. KVM - moduuli vastaa laitteistotason virtualisointituen hallinnoinnista. Nämä laitetason palvelut saatetaan ohjelmistojen käyttöön laitenoden /dev/kvm – kautta. Tätä laitetta voidaan kutsua Linuxin userspacesta ioctl-funktioiden avulla. Sen avulla voidaan:

i) Luoda uusi virtuaalikone
ii) Allokoida muistialue virtuaalikoneelle

iii) Lukea ja kirjoittaa virtuaaliprosessorin rekisterien sisältöä/sisältöihin.

iv) Laukaista virtuaaliprosessorilla keskeytys

v) Suorittaa virtuaaliprosessorin toimintaa toisin sanoen ajattaa sen prosessia (Kivity et al. 2007)

Laitetta voidaan hyödyntää siis mistä tahansa ohjelmistosta käsin, sen hallintaan ei tarvita KVM - prosessia. Liitteessä B on Intelin Jost Triplettin vuonna 2015 luoma c - kielinen esimerkki, joka luo uuden 16-bittisen x86-virtuaalikoneen, lisää sille virtuaalisen prosessorin ja määrittelee sen rekisterien sisällön, allokoi koneelle muistiavaruuden ja lopuksi käynnistää koneen. Virtuaalikoneella ei käytetä käyttöjärjestelmää, vaan prosessoria ohjataan suoraan x86-arkkitehtuurin käskyillä, jotka on määritelty muuttujaan *code*. Käskyt on kommenteissä esitetty myös assemblynä. Kyseinen virtuaaliprosessori laskee yhteen muistista löytyvät arvot 2 + 2, muuttaa sen asciiksi ja tulostaa sarjaporttiin näkyville laskusuorituksen vastauksen.

Esimerkki on hyvin yksinkertainen, mutta samaa tekniikkaa voidaan hyödyntää esimerkiksi tietoturvaohjelmistoissa ja selaimissa, kun halutaan käyttöön muusta järjestelmästä eriytetty hiekkalaatikko (engl. sandbox). Tällöin potentiaalisesti vihamielistä koodia voidaan huoletta ajaa virtuaalikoneen avulla ilman, että se pääsee vaikuttamaan alla olevaan järjestelmään millään tavalla.

Suurin osa x86 - käskykantojen laajennuksista on perinteisesti otettu käyttöön kaikkien valmistajien tuotteissa samanlaisina. Kuten edellisessä luvussa todettiin, virtualisointilaajennusten kohdalla näin ei ole kuitenkaan toimittu, vaan sekä Intel että AMD julkaisivat omat ratkaisunsa laitteistoavusteiseen virtualisointiin. Nämä ratkaisut (VT-X ja AMD-V) ovat arkkitehtuuriltaan hyvin samantyyppisiä, mutta niiden x86 – käskykantaa laajentavien käskyjen nimeämistä ei ole vakioitu. Näin ollen näitä ratkaisuja käyttävän ohjelman pitää osata hyödyntää molemmista vaihtoehdoista löytyviä käskyjä. (Kivity et al. 2007)

KVM ratkaisee tämän ongelman hyvin joustavalla tavalla. Virtualisoinnissa tarvittava perustoiminnallisuus on sijoitettu kvm.ko – moduuliin. Se sisältää myös funktio-osoitin vektorin kvm_arch_ops, joka puolestaan kutsuu laitteistoarkkitehtuurispesifisiä moduuleita kvm-intel.ko tai kvm-amd.ko tarpeen mukaan. Tämä ratkaisu mahdollistaa samalla helpon laajennuksen, mikäli myöhemmin on tarvetta tukea myös muita arkkitehtuureita.

3.1.2 Qemu-kvm

Qemu-kvm on alkuperäisestä QEMU - projektista jatkokehitetty versio, joka tarjoaa KVM - virtualisointipaketissa laitteiston emulointiin liittyvät palvelut (katso kuva 3.1). Näihin palveluihin tärkeimpinä kuuluu muun muassa:

i) Prosessorin virtualisointi. Qemu kykenee emuloimaan liitteessä A tarkemmin luetteloituja eri prosessorimalleja. Jokainen virtuaaliprosessori näkyy virtualisointialus-

tassa omana prosessinaan, joten sen toimintaa voidaan seurata ja hienosäätä esimerkiksi muuttamalla ajoprioriteettia.

ii) Virtuaalisen BIOS - piirin toiminta. Oletus - BIOSina käytetään Seabios-projektin PC BIOSia.

iii) Väyläarkkitehtuurien, kuten PCI-, PCIE- ja USB - väylien, toiminnan emulointi. KVM - virtualisointijärjestelmässä on myös mahdollista tuoda alustakoneen PCI(E) - väylään liitetty laite suoraa virtuaalikoneen käyttöön.

iv) Paikallisten massamuistilaitteiden emulointi mukaanlukien IDE-, SATA- ja SCSI-ohjaimet.

v) iSCSI-tekniikan tuki. Kyseinen tuki voidaan tarjota KVM:n puolesta suoraan alustakoneessa tai QEMU:n puolelta emuloituna.

vi) Secure Shell (ssh) levykuvien tuki. Virtuaalikoneelle voidaan tarjota levykuvatiedostoja SFTP-protokollan yli verkosta. Tämä on erittäin kätevä toiminto CD/DVD - levykuvien kanssa, sillä esimerkiksi virtuaalikoneen asennus voidaan tehdä miltä tahansa SFTP - palvelimelta Internetissä.

vii) Verkkokorttien emulointi. QEMU tarjoaa emuloitaviksi verkkokorteiksi ajureilla ne2k_isa, i82551, i82557b, i82559er, ne2k_pci, pcnet, rtl8139 ja e1000 toimivia laitteita. Nämä ovat erittäin laajasti tuettuja eri käyttöjärjestelmissä.

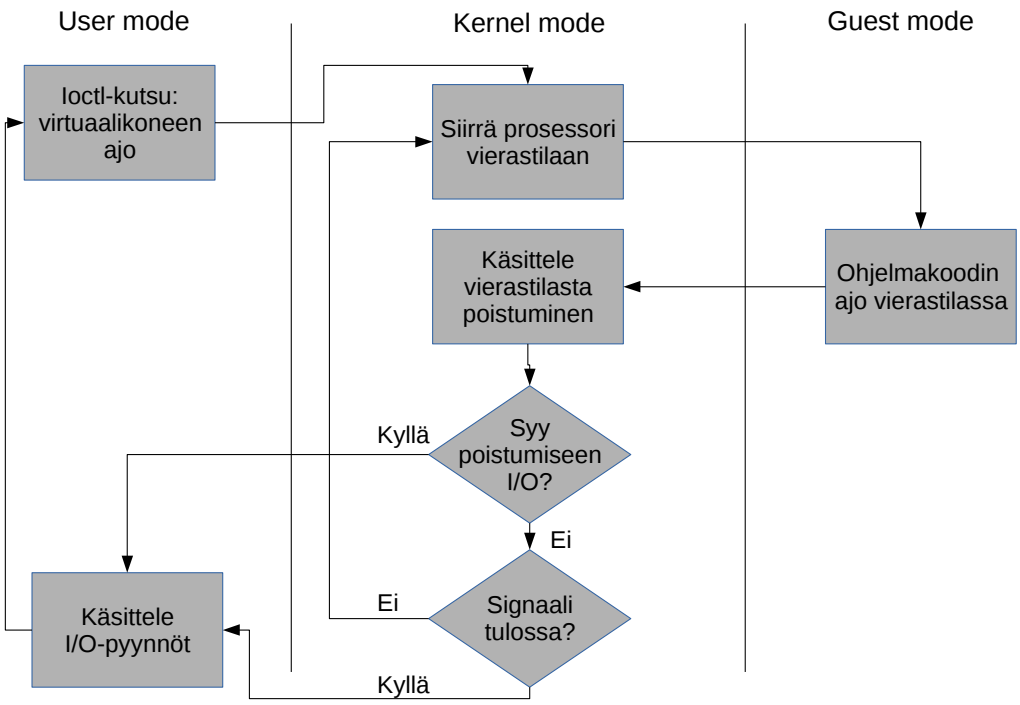
vi) Emuloidut näytönohjaimet. QEMU kykenee emuloimaan yleisesti eri käyttöjärjestelmissä tuettuja perustason näytönohjaimia. Näiden kuva voidaan välittää hallintaohjelmistolle VNC- tai SPICE - protokollien avulla.

(Red Hat 2015, Qemu 2016, Libvirt 2016)

Luvussa 5 suoritetaan mittausten avulla vertailuja QEMUn toteuttamien prosessori-, levyohjain- ja verkkokorttiemulointien osalta. Näitä emuloituja vaihtoehtoja verrataan paravirtualisoituihin VirtIO – laiteohjaimiin.

3.2 KVM:n toiminta

Kuten luvun 2 lopussa aiemmin todettiin, prosessorien käskykantojen virtualisointilaajennukset lisäävät prosessorille uuden toimintatilan nimeltään vierastila. Tätä mukaillen, KVM-moduuli lisää Linuxin ydintilan (engl. kernel mode) ja käyttäjätilan (engl. user mode) rinnalle uuden vierastilan (engl. guest mode). Virtuaalikoneita ajetaan pääsääntöisesti tässä tilassa, kunnes tapahtuu jokin poikkeus. Näihin poikkeuksiin lukeutuvat virtualisointiytimen määrittelemät käskykannan käskyt ja rekisterin käsittelyt. Myös laitteistolle menevä i/o-liikenne lukeutuu näihin poikkeuksiin. Tämä on kuvattu tarkemmin oheisessa kuvassa 3.2:



Kuva 3.2: Silmukka, jossa virtuaalikonetta suoritetaan. Kuva mukaillen (Kivity et al. 2007)

Käydään tämän kaavion silmukan toimintaa hieman tarkemmin läpi esimerkin avulla:

i) Käyttöjärjestelmä saa käyttäjätilassa toimivalta hallintaohjelmalta pyynnön, ioctl - kutsun, käynnistää virtuaalikoneen prosessorin suoritus. Ydintilassa toimiva virtualisointiydin pyytää alustakoneen prosessoria siirtymään vierastilaan, jossa tapahtuu varsinainen virtuaalikoneen koodin ajo. Ajoa jatketaan, kunnes kohdataan keskeytys, esimerkiksi i/o-laitteisiin kohdistuva käsky, kaapattavaksi määritelty käskykannan käsky tai rekisterin käsittely tai ulkopuolinen signaali, esimerkiksi verkkolaitteelle tulevaa liikennettä.

ii) Mikäli keskeytyksen syy on virtuaalikoneen suorittama i/o - käsky tai sille tuleva signaali (esimerkiksi virtuaalikoneelle tulevaa verkkoliikennettä), virtualisointiydin pyytää Linux - kerneliä käsittelemään pyynnön käyttäjätilassa. Tämän jälkeen palataan vierastilaan virtuaalikoneen ajamista varten.

iii) Mikäli keskeytyksen syy on jokin muu, virtualisointiydin käsittelee tapahtuman ja jatkaa vierastilassa virtuaalikoneen ajamista.
(Kivity et al. 2007, Rizzo et al. 2013)

3.3 Muistinhallinta

KVM - virtualisointijärjestelmässä keskusmuistin hallintaan käytettävät palvelut tulevat Linux - ytimeltä. Koska KVM - virtualisointiydin koostetaan KVM - moduulista ja Linux - ytimestä, on sen käytössä aina viimeisin Linuxiin kehitetty tekniikka.

Erityisesti KVM - järjestelmän muistinkäyttöä varten on kehitetty KSM- ja Memory Balloon – tekniikat, jotka esitellään tarkemmin alaluvuissa.

3.3.1 KSM – Kernel Same-page Merging

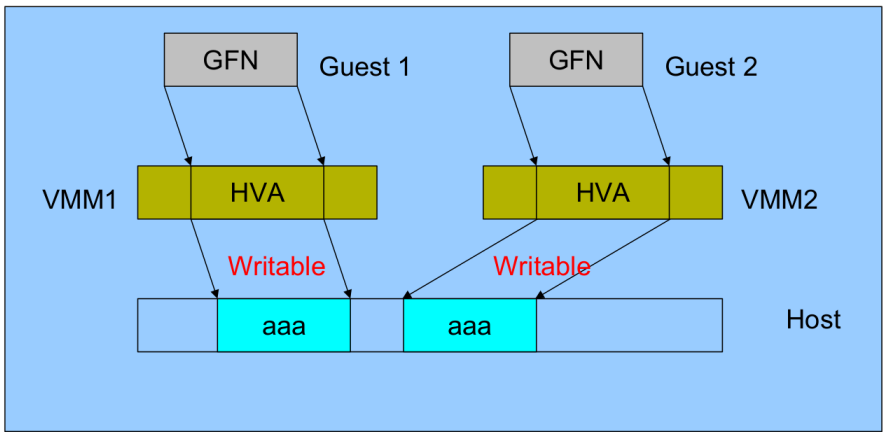
Kernel Same-page Merging (KSM) on KVM - virtualisointiytimen käyttämä tekniikka, joka mahdollistaa täysin identtisten muistialueiden jakamisen virtuaalikoneiden kesken ja siten deduplikoinnin keinoin vähentää käytetyn keskusmuistin määrää. Yleensä nämä jaetut muistialueet sisältävät virtuaalikoneille yhteisten kirjastojen tai muiden identtisten ja usein käytettävien prosessien ohjelmakoodia. Menetelmän avulla virtualisointialustassa voidaan ajaa fyysisiä muistirajoitteita suurempi määrä samantyyppisiä ja keskenään riittävän identtisiä käyttöjärjestelmiä. (Parker et al. 2015)

KSM kehitettiin alunperin KVM - virtualisoinnin käyttöön, mutta voi olla yhtä hyödyllinen mille tahansa prosessille, joka generoi keskusmuistiin samasta sisällöstä monta eri instanssia. (Eidus et al. 2009)

Jaetun muistin käsite on hyvin yleinen nykyaikaisissa käyttöjärjestelmissä. Kun uusi prosessi käynnistetään, jakaa se käynnistäneen prosessin kanssa koko muistialueensa. Kun toinen prosesseista yrittää muuttaa tämän muistin sisältöä, käyttöjärjestelmän ydin varaa sille uuden muistialueen, kopioi alkuperäisen sisällön sinne ja antaa ohjelman muokata tätä uutta aluetta. Tämä tunnetaan copy on write – menetelmänä. (Parker et al. 2015)

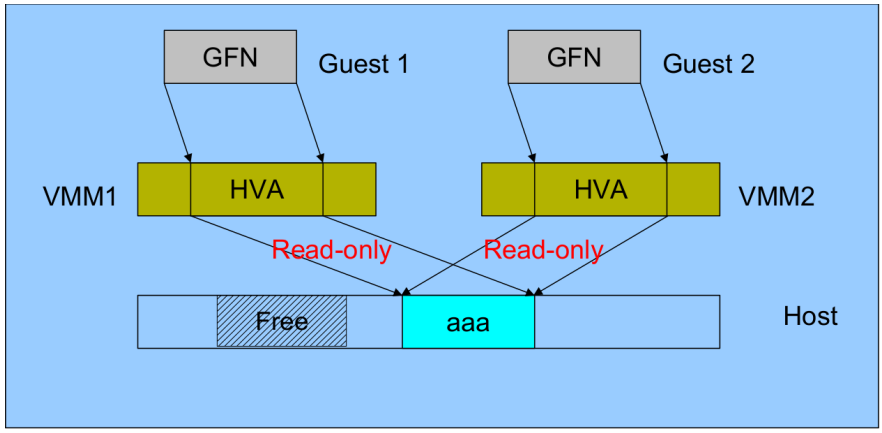
KSM on Linux - ytimen ominaisuus joka yrittää hyödyntää tätä menetelmää käänteisesti. Se antaa ytimelle mahdollisuuden tarkastella useamman käynnissä olevan prosessin muistialueita. Mikäli jotkin muistialueet tai -sivut ovat identtisiä, KSM vähentää näiden duplikaattien määrän yhteen. Tämä muistialue merkataan kirjoitettaessa kopioitavaksi, eli mikäli sen sisältöä olisi tarpeen muuttaa, luodaan muokkaavaa prosessia varten uusi muistisivu. Käydään prosessi läpi vaiheittain:

i) alkutilanteessa (kuva 3.3) virtuaalikoneiden Guest 1 ja Guest 2 muistiosoittimet GFN (engl. Guest Frame Number) viittaavat virtualisointiytimen MMU – taulussa omien muistialueidensa HVA (engl. Host Virtual Address) kautta virtualisointialustan fyysisiin muistiosoitteisiin. Molempiin muistialueisiin voi kirjoittaa dataa.



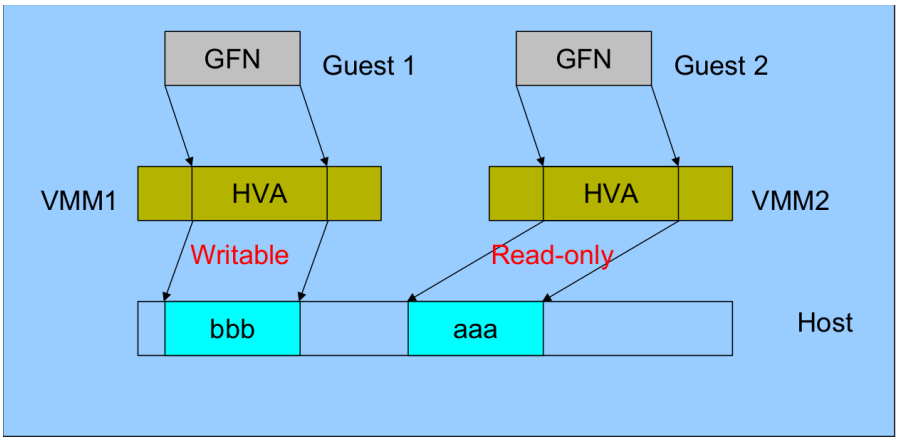
Kuva 3.3: KSM: alkutilanne ennen muistisivujen yhdistämistä. (Guangrong 2011)

ii) KSM analysoi virtuaalikoneiden käyttämien muistialueiden sisällön ja havaitsee sen yhteneväiseksi. KSM muuttaa virtuaalikoneen 1 käyttämän HVA:n viittaamaan samaan fyysiseen muistialueeseen johon virtuaalikoneen 2 HVA viittaa. Turha dataduplikaattialue keskusmuistissa merkitään vapaaksi (kuva 3.4). Yhteinen muistialue merkitään kirjoitettaessa kopioitavaksi. Virtuaalikoneet voivat siis vapaasti lukea yhteisen muistialueen sisältöä.



Kuva 3.4: KSM: Tilanne muistisivujen yhdistämisen jälkeen. Muistialue on merkitty kirjoitettaessa kopioitavaksi. (Guangrong 2011)

iii) Virtuaalikone 1 haluaa kirjoittaa muistialueelle uutta dataa (kuva3.5). Tämä kirjoitusyritys laukaisee KSM:n copy-on-write – menettelyn, jossa virtualisointiydin varaa uuden muistialueen ja suorittaa virtuaalikoneen 1 pyytämän datan kirjoittamisen sinne. Tämän jälkeen se päivittää HVA:n viittaamaan uuteen fyysiseen muistialueeseen.



Kuva 3.5: KSM: Tilanne muistialueen sisällön muuttamisen jälkeen. (Guangrong 2011)

iv) Virtuaalikoneen 2 muistialue jää read-only – tilaan, ja merkitään kirjoitettavaksi vasta tarvittaessa. Tämän tarkoituksena on nopeuttaa toimintaa, mikäli KSM:n pitäisi uudelleen yhdistää muistialueet. (Guangrong 2011, Parker et al. 2015)

Tämä toiminto on virtualisoinnissa käytännöllinen. Kun virtuaalikone käynnistetään, se ainoastaan perii qemu-kvm – prosessin muistialueen. Käynnissä olevan koneen muistisisältö puolestaan voidaan jakaa muiden virtuaalikoneiden kanssa. Jako ei kuitenkaan tapahdu automaattisesti, vaan KVM:n pitää pyytää KSM:ä tutkimaan, mikäli sen käyttämillä muistialueilla on käytössä identtisiä muistisivuja ja tarvittaessa jakamaan ne. Tämä tapahtuu kutsumalla KSM:n järjestelmäkutsua *int madvise(addr, length, MADV_MERGEABLE)*. Mikäli KVM haluaa palata takaisin jakamattomiin muistisivuihin, se voi kutsulla *int madvise(addr, length, MADV_UNMERGEABLE)* peruuttaa sivunjaon. Muuttujan addr tilalle annetaan muistiosoitteen alku ja lengthin tilalle sen laajuus. Äärimmäisissä tilanteissa keskusmuisti voi jopa loppua kesken, mikäli suuri määrä muistisivuja yhtäkkiä muutetaan takaisin jakamattomiksi. Yhteisten muistisivujen etsiminen on kuitenkin prosessoria kuormittava toimi, joten toiset Linux - jakelupaketit ovat ottaneet sen oletuksena pois käytöstä. (Eidus et al. 2009)

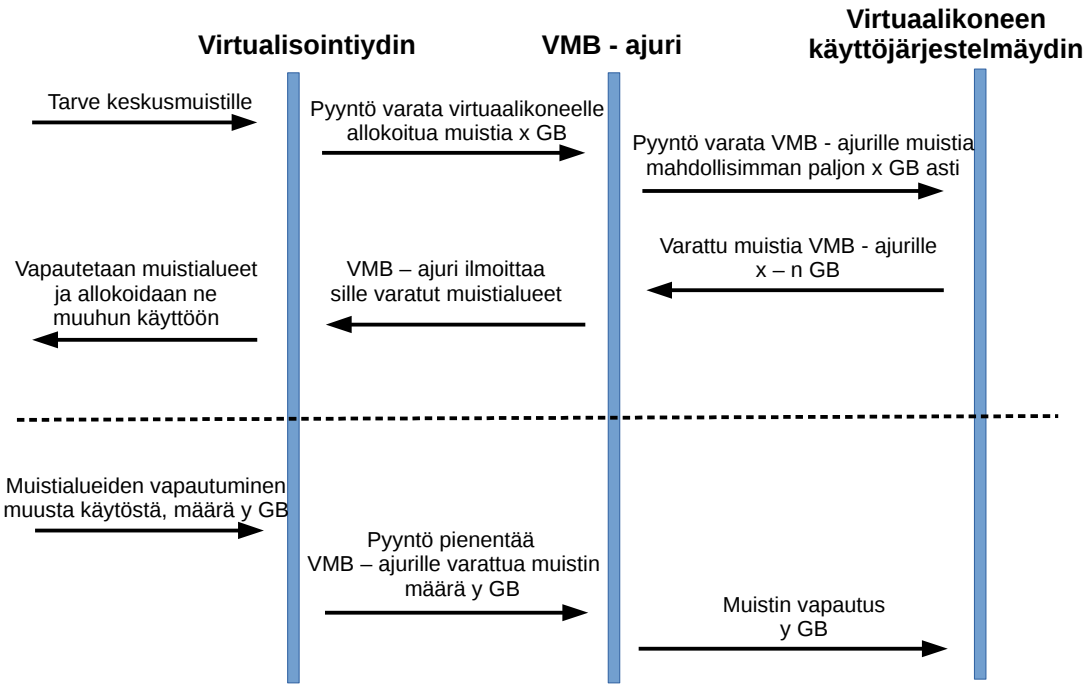
KSM ei vain vähennä keskusmuistin käyttöä, vaan myös parantaa sen suorituskykyä. Sen avulla prosessien identtinen data varastoidaan välimuistiin. Tämä parantaa virtuaalikoneiden välimuistituksen osumatarkkuutta, jolla saattaa olla tilanteesta ja käytöstä riippuen positiivinen vaikutus virtuaalikoneiden suorituskykyyn. (Parker et al. 2015)

KSM:n hallinta on toteutettu kahdella erillisellä metodilla: ksm – palvelu käynnistää ja tarvittaessa pysäyttää ytimen KSM - säikeen. Sitä hallinnoi ja hienosäätää ksmtuned – palvelu sen mukaan onko muistinjako tarpeen vai ei. Nämä palvelut asentuvat qemu-kvm - paketin mukana. Kun ksm - palvelu ei ole käynnissä, ytimen KSM-toiminto jakaa oletuksena virtuaalikoneiden kesken ainoastaan 2000 muistisivua. Tämä matala oletusarvo tarjoaa melko rajoitetusti muistinsäästöä. Palvelun ollessa käynnissä jaetaan oletuksena korkeintaan puolet virtualisointialustan muistimäärästä. (Parker et al. 2015)

Linux - ytimen KSM - toiminton asetukset ja statistiikka löytyy hakemistosta /sys/kernel/mm/ksm/. KSM:n tuen voi ottaa qemu-kvm:ssä käyttöön tiedostosta /etc/default/qemu-kvm.

3.3.2 Memory Balloon driver

KVM – virtualisointialustojen keskusmuistin hallintaan on tarjolla myös Memory balloon driver – ajuri, jolla voidaan jakaa virtuaalikoneiden käyttöön enemmän muistia kuin alustakoneessa on käytössä. Tämä toiminta nojaa virtualisointiytimen ja virtuaalikoneiden väliseen yhteistyöhön, jota ohjataan virtuaalikoneilla olevan VirtIO – pakettiin kuuluvan Virtio memory balloon – ajurin (VMB) kautta. Edellytyksenä on, että alustan virtuaalikoneet eivät käytä kaikkea niille varattua keskusmuistia. (IBM 2012)



Kuva 3.6: Kaaviossa on esitetty Virtual memory balloon - ajurin toiminta yhteistyössä virtualisointiytimen ja virtuaalikoneen käyttöjärjestelmäytimen kanssa.

Toimintaa ohjataan seuraavasti (kuva 3.6):

i) Virtualisointiydin lähettää virtuaalikoneelle pyynnön vapauttaa tietty määrä keskusmuistia takaisin sen omaan käyttöön

ii) Virtuaalikoneen VMB – ajuri vastaanottaa pyynnön, ja pyrkii varaamaan itselleen pyynnön verran lisää muistia virtuaalikoneen muistiavaruudesta. Mikäli muistia ei voida varata pyydettyä määrää, VMB varaa suurimman mahdollisen määrän.

iii) VMB ilmoittaa virtualisointiytimelle varaamansa muistialueet

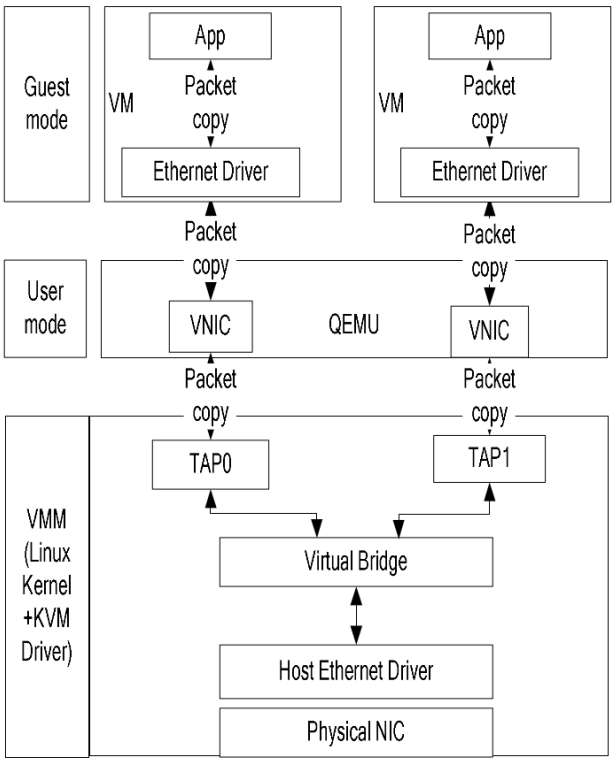
iv) Virtualisointiydin vapauttaa tätä virtuaalista muistialuetta vastaavat oikeat muistialueet, ja jakaa ne muuhun käyttöön.

v) Mikäli muistialueet myöhemmin vapautuvat tästä muusta käytöstä, virtualisointiydin lähettää VMB – ajurille pyynnön vapauttaa muistia takaisin virtuaalikoneen käyttöön.
(IBM 2012, Herrmann et al. 2016)

VMB on KSM – tekniikkaa turvallisempi tapa säästää virtualisointialustan keskusmuistia. Sen käyttö ei mahdollista muistin ylivaraamista, joten se ei myöskään aiheuta keskusmuistin loppuessa tarvetta siirtää muistisivuja käyttöjärjestelmän tarjoamaan hitaampaan näennäismuistiin (swap). (Herrmann et al. 2016)

3.4 Virtuaalikoneiden tietoliikenne

Linux tarjoaa tuen IEEE 802.1d – standardin mukaiselle verkkosillalle, joka kytkee toisiinsa kaksi verkkosegmenttia OSI – mallin siirtokerroksella. Tätä voidaan käyttää jakamaan virtualisointialustan fyysinen verkkokortti virtuaalikoneiden kanssa. Koska silta toimii layerillä 2, kaikki verkkoprotokollat voivat kulkea sen kautta virtuaalikoneille. Tämä kytkentätapa on myös IBM:n suosittelema best practice KVM:n verkkojen virtualisoimiseen (IBM 2012). (Huffman et al. 2016, IEEE 802.1QTM-2014/Cor)



Kuva 3.7: KVM:n verkkoliikenteen I/O - arkkitehtuuri (Li et al. 2009)

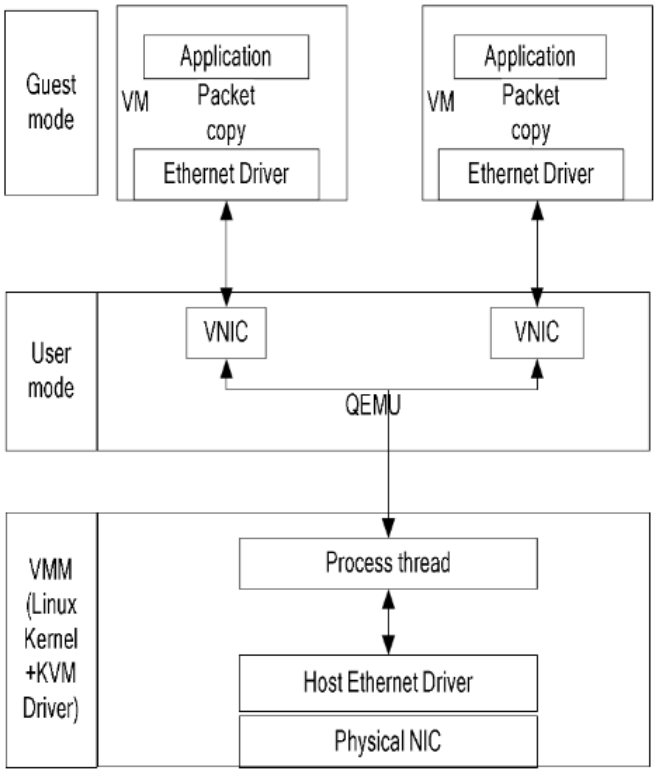
Kuva 3.7 kuvaa KVM:n verkkoliikenteen I/O – arkkitehtuurin. Siinä virtualisointialustan fyysinen verkkoliitäntä on kytkettynä Linuxin verkkosiltaan. Jokaista virtuaalikoneen virtuaalista verkkokortti kohti on olemassa yksi virtualisointiytimen TAP – laite, joka toimii sillan tavoin siirtokerroksella. Ne toimivat Linuxin ydintilassa ja ovat kytkettynä siltaan. Qemu emuloi käyttäjätilassa virtuaalikoneiden verkkokorttien toimintaa. Virtuaalikoneiden ytimet keskustelevat laiteajurin valityksellä tämän emuloidun verkkokortin kanssa. (Li et al. 2009, Huffman et al. 2016)

Kun virtuaalikoneelle saapuu verkkoliikennettä, ohjataan se MAC – osoitteen perusteella verkkosillalla olevalle oikealle TAPille. Koska tämä kaikki tapahtuu ydintilassa, paketin sisältöä ei tarvitse kopioida eteenpäin muistissa. TAP – laitteen jälkeen tilanne onkin toinen, ja paketin sisältämä data kopioidaan käyttäjätilassa toimivalle Qemulle. Toinen kopiointi tapahtuu, kun data välitetään virtuaalikoneen käyttöjärjestelmäytimelle. Paketti kopioidaan vielä kolmannen kerran muistissa kun virtuaalikoneen ydin välittää sen asianmukaiselle ohjelmalle, joka toimii virtuaalikoneen omassa user spacessa. (Li et al. 2009)

Edellä olevasta verkkoliikenteen analyysistä huomataan, että vastaanotettu ja virtuaalikoneelle menevä paketti käy läpi kaksi eri virtuaalista verkkokorttia ja kolme keskusmuistissa tapahtuvaa kopiointia. Ketju on täsmälleen sama myös lähetettävien pakettien osalta. Tämä pitkä I/O – ketju lisää paketin saapumiseen viivettä ja siten myös hi-

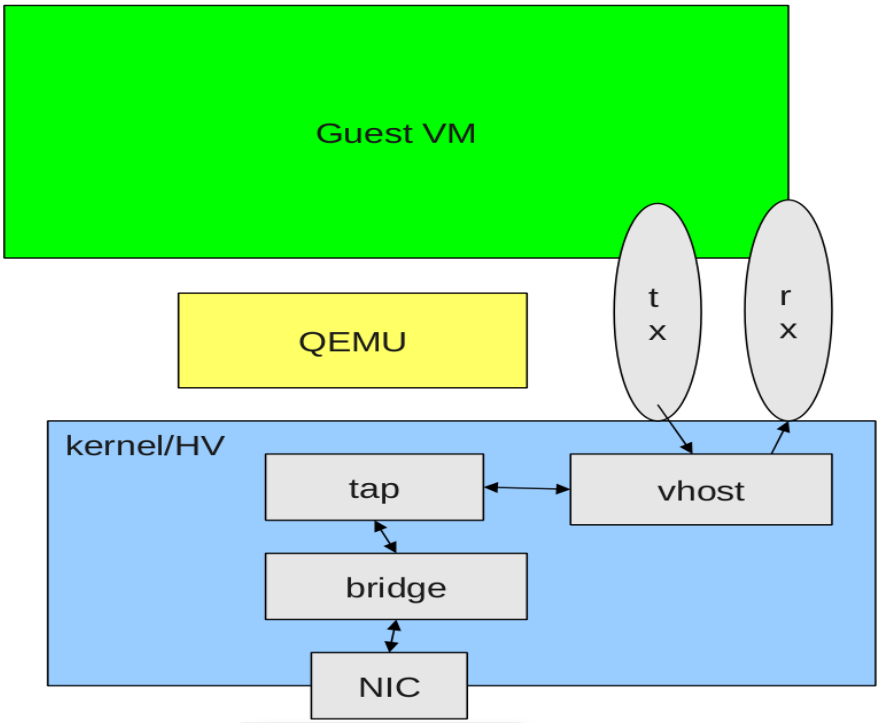
dastaa verkkoyhteyden toimintaa. Tämä kuluttaa myös virtualisointialustan CPU – aikaa. (Li et al. 2009)

Ongelmaa parantamaan on ehdotettu menetelmää, jossa virtualisointiytimeen lisätään prosessisäie, joka vastaanottaa kaikki fyysiselle verkkokortille saapuvat paketit ja toimittaa ne suoraan QEMU:n virtualisoimille verkkokorteille. Tällöin virtuaalikoneen ydin voisi lukea ne suoraan virtuaaliverkkokortilta ja tarvittaisiin vain yksi kopiointi virtuaalikoneen kernel spacesta user spacessa toimivalle ohjelmistolle (kuva 3.8)



Kuva 3.8: Verkkoarkkitehtuurin nopeuttamiseksi ehdotettu vaihtoehto, jossa virtualisointialustan kernel modessa toimiva säie vastaanottaa saapuvat paketit ja kopioi ne suoraan virtuaaliverkkokortille. (Li et al. 2009)

Laajempaan käyttöön on kuitenkin valikoitunut vhost-net – ratkaisu. Vhost-net on käyttöjärjestelmäytimen tasolla toimiva ohjelmistokomponentti. Se tehostaa verkkovirtualisoinnin toimintaa siirtämällä virtuaalikoneiden verkkokorttien pakettien prosessoinnin user spacessa toimivalta QEMU – prosessilta virtualisointiytimeen vhost-net – ajurille. Tämä vähentää tarvittavien keskusmuistikopiointien määrää yhdellä. Samalla se pienentää pakettien käsittelyn aiheuttamaa latenssia. Vhost-netin toiminta on esitetty kuvassa 3.9. (Burns 2010)



Kuva 3.9: Vhost-net - modulin toiminta. Verkkoliikenteen käsittely on siirretty pois user spacesta toimivalta QEMU - prosessilta ja siirretty virtualisointiytimen vhost – moduliin. (Burns 2010)

Mikäli virtualisointialustalla vhost-net – moduuli on ladattuna, käytetään sitä oletuksena kaikkien virtio – verkkolaitteiden kanssa. Mikäli sitä ei jostakin syystä haluta käyttää, voidaan vhost-net poistaa käytöstä yksittäiseltä virtuaalikoneelta käyttämällä oletusajurin sijaan Qemu – ajuria. Tämä tapahtuu määrittelemällä ajuri käyttöön virtuaalikoneen XML – tiedostossa:

```
<interface type="network">
  ...
  <model type="virtio"/>
  <driver name="qemu"/>
  ...
</interface>
```

Asetus on verkkokorttikohtainen. Koska vhost-net – ajuri toimii vain virtio – laitteiden kanssa, käytetään emuloitujen verkkokorttien kanssa edelleen kuvan 3.7 kuvaamaa prosessia. (Russell 2008, Burns 2010)

3.5 Tallennusratkaisut

KVM on erittäin kaikkiruokainen tallennustilan suhteen. Virtualisointijärjestelmään itseensä voidaan osoittaa massamuistia hyvin laajalta skaalalta, mutta valikoima laajenee vielä dramaattisesti kun huomioidaan Linuxin tarjoama tallennustilatuki. Tallennusratkaisut voidaan kuitenkin listata kolmeen KVM:n käyttämään päätyyppiin, joita ovat:

i) Lohkotason laitteet: paikalliset kiintolevyt, RAID – pakat, LVM – osiot, käytännössä kaikki Linuxin tukemat suorakytkentäiset tallennusjärjestelmät (DAS) ja SAN – järjestelmät

ii) Tietyt tiedostotason järjestelmät, kuten NFS- , FTP- ja SFTP – jaot.

iii) Levykuvat tiedostojärjestelmässä, kuten QCOW2, RAW, VMDK, ISO, QED

iv) virtualisointialustan tallennusjärjestelmien hakemistot. Hakemistojen käyttö ei kuitenkaan ole suositeltavaa, sillä virtualisointiytimellä käytettävät ohjelmistot voivat käsitellä niiden sisältöä virtuaalikoneen siitä tietämättä (Herrmann et al. 2016).

(IBM 2012, Libvirt 2016)

Lohkotason laitteita ja levykuvia voidaan käyttää virtuaalikoneille näkyvänä paikallisena tallennustilana toisin sanoen siis virtuaalikoneiden kiintolevyinä. Lohkotason laitteiden suorituskyky on yleensä kuitenkin parempi verrattuna levykuvatiedostoihin. Levykuvat tarjoavat kuitenkin monta muuta etua, kuten:

i) Sijoitettavuutta. Levykuvat on helpompi sijoittaa massamuistilaitteille kuin esimerkiksi osiot.

ii) Käytettävyyttä. Tiedostojen ylläpito on helpompaa kuin levyjen, osioiden, LVM – osioiden tai levypakkojen hallinta.

iii) Siirrettävyys. Tiedostoja on helppo siirtää eri sijaintien tai laitteiden välillä.

iv) Kloonaus. Tiedosto on helppo kloonata toisen virtuaalikoneen käyttöön.

v) Thin provisioning, jossa tilaa varataan levykuvalta ja sen käyttämältä tallennusvälineeltä vain tarvittava määärä, vaikka virtuaalinen kapasiteetti olisi suurempikin. Tämä mahdollistaa myös tallennustilan ylivaraamisen (over-committing), jossa levykuvien nimelliskapasiteetti on suurempi kuin tallennuslaitteella käytettävissä oleva levytila. Menetelmässä on kuitenkin riskinsä, kesken loppuva levytila aiheuttaa ongelmia virtuaalikoneilla, varsinkin jos virtualisointijärjestelmän mukaan levykuvalla pitäisi olla kapasiteetti vielä käytettävissä.

vi) Tallennus verkkoon. Levykuvat voidaan tallentaa mille tahansa verkkolaitteelle, jonka tarjoama levytila on alustakoneen käytettävissä.

Levykuvat, ja erityisesti RAW-tiedostot, ovatkin sekä IBM:n että Red Hatin suosittelemia best practice – ratkaisuja virtualisointiin. (IBM 2012, Parker et al. 2015)

Levykuvat mahdollistavat myös tilannekuvien (engl. Snapshot) ottamisen virtuaalikoneiden kiintolevyistä. Tällöin pitää käyttää tilannekuvia tukevaa levykuvaformaattia, joista käytetyin lienee qcow2. Tilannekuvat luodaan Redirect-on-Write – periaatteen mukaisesti, eli alkuperäiseen levykuvaan nähden tehdyt muutokset kirjoitetaan omaan tiedostoonsa. Pahimmillaan tämä vie kuitenkin tilaa kaksi kertaa alkuperäisen levykuvan verran. Tilannekopioita voidaan poistaa ja yhdistää tarpeen mukaan, jolloin virtualisointijärjestelmä kirjoittaa tilannekuvaan tehdyt muutokset takaisin levykuvaan. (IBM 2012)

Käytetään alustana sitten levykuvaa tai lohkotason laitetta, virtuaalikoneelle näkyvä virtuaalikiintolevy voidaan ottaa käyttää joko emuloituna tai paravirtualisoituna. Emulaation kanssa tuetaan IDE, SCSI ja SATA – väyliä. Paravirtualisoituna käytössä on Virtio – paketin Virtio-blk – ohjain. Paravirtualisoitu laite on merkittävästi emuloitua laitetta nopeampi. (Russell 2008, IBM 2012, Red Hat 2015)

4 KVM:N KÄYTTÖÖNOTTO

Tässä luvussa kerrotaan, miten KVM:n käyttöönotto onnistuu ja mitä vaatimuksia se asettaa laitteistolle. Esimerkin vuoksi asennetaan minimaalinen suositeltava ohjelmistovalikoima virtualisointialustan käyttöä varten. Samalla esitellään KVM:n yleisimpiä ja hyödyllisimpiä ominaisuuksia.

KVM - ohjelmistoa varten on luotuna valtava määrä erilaisia käyttöliittymiä niin graafisena, tekstipohjaisena kuin WWW-sivuna. Libvirt on yksi näistä hallintakäyttöliittymistä, joka tarjoaa yksinkertaisen komentorivipohjaisen käyttöliittymän KVM – ohjelmistoa varten. Se esitellään tarkemmin omassa aliluvussaan. Myös monet pilvipalveluohjelmistot hyödyntävät sitä sekä KVM – prosessin omia hallintamahdollisuuksia tarjoten asiakkaille kuitenkin graafisen käyttöympäristön. Tässä työssä ei kuitenkaan perehdytä eri käyttöliittymävaihtoehtoihin, vaan rajaudutaan KVM:n käyttöön Libvirt - ohjelmiston komentorivityökalujen kautta. Näin käyttöesimerkit ovat helposti ymmärrettävissä ja sovellettavissa kunkin käyttäjän suosimaan käyttöliittymään.

4.1 Virtualisointialustan laitevaatimukset

KVM on laitteistopohjaista virtualisointia hyödyntävä virtualisointialusta, joten alustan prosessorien pitää tukea virtualisointia. Tällä hetkellä toteutettuna on tuki Intelin VT-x – ja AMD:n AMD-V – laajennuksille. Mikäli halutaan hyödyntää PCI - läpivientiä, prosessorilta vaaditaan lisäksi IOMMU - tuki (AMD) tai VT-d – tuki (Intel). (Burns 2010)

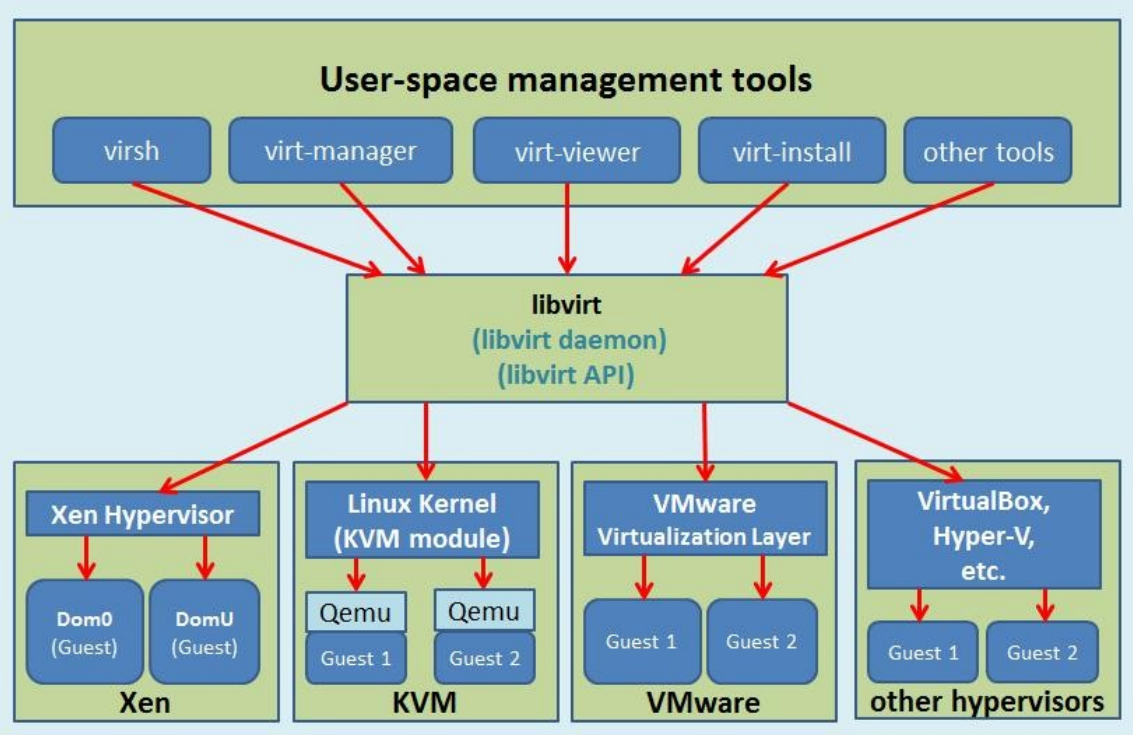
Virtualisointialustan keskusmuistin ja tallennustilan tarpeeseen vaikuttaa luonnollisesti ajettavien virtuaalikoneiden konfiguraatio. Itse alusta toimii hyvin jo 1GB keskusmuistilla ja tallennustilalla. Tyypillisesti Linux - virtuaalikoneille on hyvä varata minimissään 1GB keskusmuistia ja Windows-virtuaalikoneille puolestaan 4GB. Palvelinkäytössä Windows-käyttöjärjestelmälle on tyypillisesti hyvä tarjota vähintään kaksi virtuaaliprosessoria käyttöön.

4.2 Hyödylliset ja tarvittavat aputyökalut

Virtualisointialustan ja -resurssien hallinnoinnissa käytetään apuna useaa eri avoimen lähdekoodin ohjelmistopakettia, jotka laajentavat virtualisointiytimen sovelluspinossa ylöspäin tarjoamia toimintoja. Tässä aliluvussa on esitelty näistä tyypillisimmät vaihtoehdot.

4.2.1 Libvirt ja virsh

Libvirt on avoimen lähdekoodin ohjelmistopaketti ja -kirjasto, jolla voidaan hallita useampaa eri virtualisointijärjestelmää. Tuettuna on tällä hetkellä ainakin KVM, Xen, Vmware ESX, Qemu ja LXC. Sen ensimmäinen versio julkaistiin vuonna 2005. Libvirt tarjoaa virtualisointijärjestelmistä riippumattoman virtualisointiAPIn, jonka avulla voidaan ohjata tuettujen virtualisointijärjestelmien toimintaa. (Ashley et al. 2016)



Kuva 4.1: Libvirt - ohjelmiston tukemat virtualisointijärjestelmät ja asiakasohjelmat (Ashley et al. 2016)

Libvirt pyrkii tarjoamaan eri virtualisointialustoille yhteisen ja vakaan rajapinnan, jonka kautta virtuaalikoneita voidaan turvallisesti hallita. Koska virtuaalikoneet sijaitsevat yleensä etäyhteyksien takana, libvirt tarjoaa mahdollisuuden etäkäyttöön monella eri protokollalla. Suosittu vaihtoehto on kuitenkin tunneloida libvirtin tarjoaman rajapinnan liikenne salatun SSH – tunnelin yli. (Ashley et al. 2016)

Libvirt on suunniteltu toimimaan osana korkeamman tason hallintaohjelmia. Sitä käytetäänkin laajasti virtualisointipalveluja tuottavien pilvipalveluiden hallintaan. Koska Libvirt – kirjastot sijaitsevat yksittäisillä virtualisointialustoilla, on se suunniteltu pääsääntöisesti yksittäisen alustan hallintaan, joskin migraatiotoiminnot ylittävät tämän rajan ja tekevät yhteistyötä useamman virtualisointialustanoden kanssa. Ohjelmisto on suunniteltu valvomaan virtualisointialustan laitteistoa ja tarjoamaan tietoa sen toiminnasta. (Ashley et al. 2016)

Virsh – työkalu tarjoaa komentorivitason käyttöliittymän Libvirtin palveluiden ohjaamiseen. Sillä voidaan hallita virtualisointialustalle ajettavien virtuaali-instanssien hallintaa sekä valvoa niiden toimintaa. Työkalulla voidaan myös hallita virtualisointialustan tarjoamia palveluita, kuten tietoliikenneverkkoja ja tallennustilaa tarjoavia tallennuspooleja. Se on samalla paketin kattavin hallintatyökalu, jonka ominaisuudet kasvavat yhdessä libvirt – rajapinnan tarjoamien palveluiden kanssa. (Jones 2012, Ashley et al. 2016)

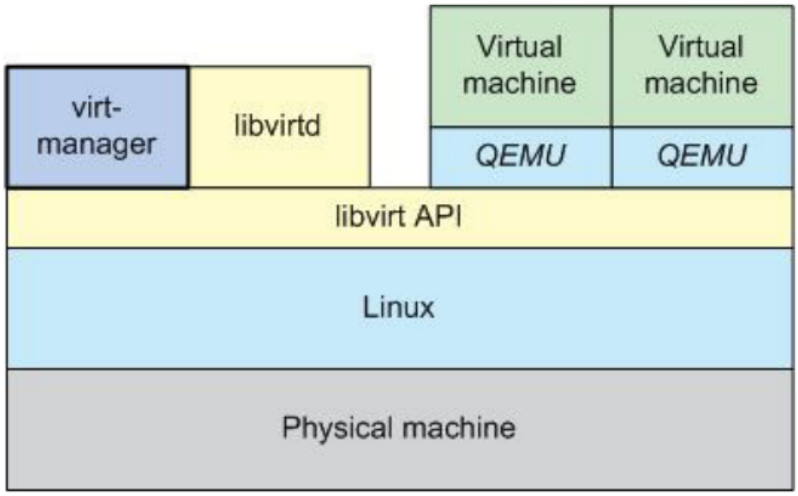
Virt-viewer on minimalistinen komentorivipohjainen työkalu, jota käytetään yksittäisen virtuaalikoneen etäkäyttöön. Se välittää virtuaalikoneen näkymän käyttäjälle, sekä välittää käyttäjän syötteen virtuaalikoneelle. Ohjelma käyttää tätä varten vaihtoehtoisesti joko VNC- tai SPICE – protokollaa. (Jones 2012)

Virt-manager on graafinen hallintaohjelma libvirt – rajapintaa varten, jota voidaan käyttää Virt-viewerin tavoin virtuaalikoneiden etäkäyttöön. Se ei ainoastaan mahdollista kuvan ja syötteen välittämistä, vaan sillä voidaan myös hallita virtualisointipalvelimen virtuaalikoneiden kokoonpanoja, verkkoyhteyksiä sekä esimerkiksi käynnistää, pysäyttää ja sammuttaa virtuaalikoneen ajo. Ohjelman avulla voidaan myös käynnistää virtuaalikoneen siirto virtualisointialustalta toiselle, kloonata virtuaalikoneesta toinen kopio käytettäväksi tai poistaa virtuaalikone kokonaan käytöstä. (Jones 2012)

Virt-install on komentorivipohjainen virtuaalikoneiden asennustyökalu. Sillä voidaan luoda libvirt – rajapintaa käyttäen uusia virtuaalikoneita. Työkalu tarjoaa erittäin laajan komentorivioptioiden tuen uusien virtuaalikoneiden asetusten konfigurointia varten. Tämä mahdollistaa automatisoinnin, jonka avulla uusi virtuaalikone voidaan asentaa alusta loppuun täysin ilman käyttäjän valvontaa. (Jones 2012)

Virt-image on toiminnallisuudeltaan ja käyttötarkoitukseltaan hyvin vastaava kuin Virt-install. Se lukee kuitenkin virtuaalikoneen muodostamiseen käytetyn syötteen komentorivin sijaan XML – tiedostosta. (Jones 2012)

Virt-clone on komentorivipohjainen työkalu, joka helpottaa virtuaalikoneiden kloonausta. Vaikka kopion virtuaalikoneesta voi tehdä myös Linuxin peruskomennoilla, lähinnä kopioimalla XML – määritystiedosto ja levyimage, helpottaa virt-clonen käyttö prosessia vaihtamalla automaattisesti virtuaalikoneelle uniikin uuid:n ja verkkokorttien MAC – osoitteet kloonauksen myötä ristiriitojen välttämiseksi. Esimerkki näistä arvoista nähdään aliluvussa 4.4 määritellyssä XML – tiedostossa. (Jones 2012)



Kuva 4.2: Libvirt - hallintatyökalujen protokollapino. Kuvassa Linux käsittää KVM:n tapauksessa KVM - moduulin ja Linux - ytimen muodostaman virtualisointialustan. (Jones 2012)

Kuvassa 4.2 on kuvattuna Libvirt – hallintatyökalujen protokollapino. Edellisissä kappaleissa kuvatut hallintaohjelmistot toimivat Linuxin käyttäjätilassa. Libvirt API toimii rajapintana hallintaohjelmiston, virtualisointiytimen ja niin ikään myös käyttäjätilassa ajettavien virtuaalikoneiden QEMU – prosessien kesken. Esimerkiksi kun käyttäjä haluaa sammuttaa virtuaalikoneen, välitetään tieto hallintaohjelmistolta libvirtille, joka puolestaan pyytää virtualisointiydintä sammuttamaan virtuaalikoneen hallitusti. Tämän jälkeen virtualisointiydin tuhoaa ajettavan virtuaalikoneen cpu - prosessit. (Jones 2012)

4.2.2 Bridge-utils, OpenSSH-server sekä valvontatyökalut

Bridge-utils – paketti toteuttaa Linuxissa IEEE 802.1d – standardin mukaisen verkkosillan, jonka avulla virtuaalikoneiden verkkokortit kytketään virtualisointialustan fyysiseen verkkokorttiin. Tämän sillan toimintaa on käsitelty tarkemmin jo aiemmassa aliluvussa 3.4. (Huffman et al. 2016)

OpenSSH-server on turvalliset etäyhteydet tarjoava ohjelmistopaketti. Sen kehitystön on aloittanut Tatu Ylönen vuonna 1999. KVM-virtualisoinnissa turvallinen etäyhteys virtualisointialustaan on tärkeä, ja sen takia hallintayhteyksien liikenne tunneloidaan SSH-putken läpi.

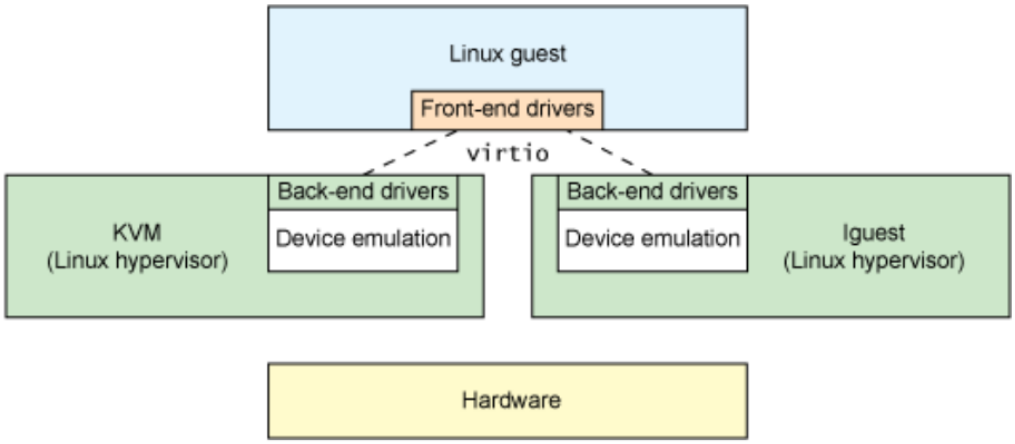
Linux tarjoaa KVM:n käyttöön kattavat valvontatyökalut. Koska virtuaalikoneiden prosessorin näkyvät virtualisointialustalla normaaleina prosesseina, voidaan niiden toimintaa tarkastella esimerkiksi työkalujen ps, htop ja Virt-top avulla. Ne kaikki kertovat tarkempaa tietoa prosessien tilasta. Jälkimmäiset kaksi ohjelmaa tarjoavat komentoriviltä käytettävän merkkipohjaisen valvontajärjestelmän prosessien tarkkailuun. (Red Hat 2015, Herrmann et al. 2016)

Verkkolaitteiden valvontaan ja hallintaan on tarkoitettu ohjelmistot ethtool ja brctl. Ethtoolin avulla voidaan vaikuttaa verkkokorttien toimintaan. Brctl – ohjelmalla voidaan puolestaan hallinnoida Linuxin verkkosiltoja ja muuttaa niiden toimintaa tarpeen mukaan. Sillä voidaan esimerkiksi ottaa käyttöön STP – protokolla tai listata sillataan kuuluvat fyysiset ja virtuaaliset verkkolaitteet. (Herrmann et al. 2016)

4.2.3 Virtio

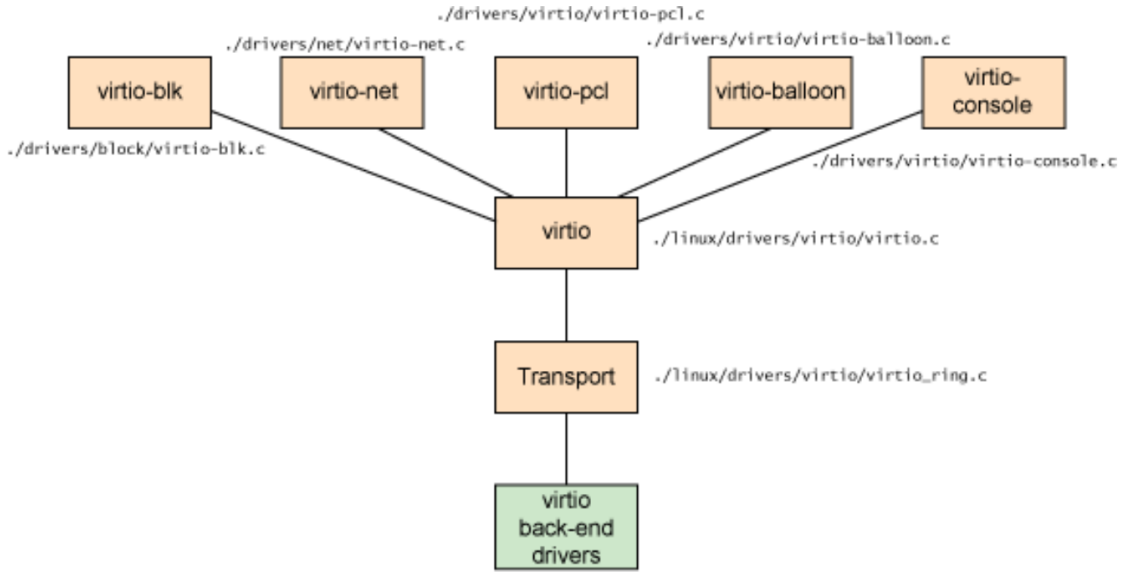
Tässä aliluvussaan esitellään ohjelmistopaketin rakenne ja sen toimintaperiaate. Virtion yksittäisiin komponentteihin on tutustuttu tarkemmin luvussa 3 ja luvussa 5 esitellään niiden vaikutusta virtualisoinnin suorituskykyyn.

Virtio on ohjelmistopaketti, joka tarjoaa abstraktiokerroksen paravirtualisoiduille laiteajureille. Sen kehityksen on aloittanut Rusty Russell. Kuvassa 4.3 nähdään Virtio – ajuripaketin komponenttien yleinen rakenne. Ajurit koostuvat kernel spacen sisällä toimivasta back-end – ajurista sekä user spacessa ajettavasta virtuaalikoneen Front-end – ajurista:



Kuva 4.3: Virtio - ajuripaketin komponettien rakenne. (Jones 2010)

Ajurit kommunikoivat keskenään Virtio API:n kautta. (Russell 2008, Jones 2010)



Kuva 4.4: Kaikki Virtio:n tarjoamat laiteajurit. (Jones 2010)

Virtio tarjoaa useamman eri paravirtualisoidun laiteajurin virtuaalikoneiden käyttöön. Ajurit ovat kuvattuna kuvassa 4.4. Virtio-blk ajuri tarjoaa tuen lohkotason (engl. Block level) laitteiden käyttöön. Virtio-net on verkkokorttiajuri. Virtio-pci emuloi PCI – laitteita. Virtio-balloon toteuttaa puolestaan palvelun, jolla virtuaalikoneelle osoitettua muistia voidaan palauttaa takaisin virtualisointialustan käyttöön. (Russell 2008, Jones 2010)

4.3 Virtualisointialustan asennus ja konfigurointi

Virtualisointialustan asennus tehdään tässä työssä Ubuntu 16.04 LTS server – jakelupakettia käyttämällä. Ubuntu on Debian-jakelun unstable - haaraan pohjautuva jakelupaketti. LTS - versioon tarjotaan tukea viiden vuoden ajan.

Ennen virtualisointilaajennusten asentamista on hyvä tarkistaa alustakoneen prosessorin virtualisointilaajennusten tuki. Tämä onnistuu komennolla:

egrep "vmx|svm" /proc/cpuinfo

Virtualisointijärjestelmä voidaan asentaa käyttöön asentamalla seuraavat paketit ja niiden vaatimat paketit:

apt-get install qemu-kvm bridge-utils libvirt-bin ethtool htop virt-top

Asennus muuttaa Linux - ytimen virtualisointiytimeksi lisäämällä siihen virtualisointilaajennusten käytön mahdollistavan KVM – moduulin. Samalla asennetaan myös

virtualisointialustan hallintaan tarkoitettu Libvirt – paketti sekä joukko muita virtualisointialustalla hyödyllisiä ohjelmistoja.

Määritellään verkkoasetukset hakemistoon /etc/network/interfaces:

```
auto em2
    iface em2 inet manual

auto br-vlan630
    iface br-vlan630 inet static
    address 37.24.12.27
    network  37.24.12.0
    netmask 255.255.255.0
    gateway 37.24.12.1
    dns-search local.testdomain.fi
    dns-nameservers 8.8.8.8 8.4.4.4
    bridge_ports em2
    bridge_fd 9
    bridge_hello 2
    bridge_maxage 12
    bridge_stp off
```

Tämä luo sillan br-vlan630, joka käyttää alustakoneen integroitua verkkokorttia em2 fyysisenä verkkolinkkinä. Virtuaalikoneiden verkkokortit voidaan yhdistää tähän samaan siltaan. Silta itsessään toimii OSI-mallin siirtokerroksella, mutta koska sille on määritelty myös oma staattinen IP – osoite, näkyy se virtualisointialustalle myös verkkokerroksen laitteena toisin sanoen verkkokorttina. (Linux Foundation 2009) (IEEE 802.1QTM-2014/Cor)

IP – osoitteen määrittely on kuitenkin täysin valinnaista, eikä se vaikuta virtuaalikoneiden sillan kautta suorittamaan verkkoliikennöintiin. Se kuitenkin avaa reitin virtualisointialustalle siltaan kytketystä verkosta, joten tämä pitää huomioida alustan tietoturvasta huolehdittaessa. (Parker et al. 2015)

Sillalle br-vlan630 määritelty asetus bridge_stp off poistaa STP-protokollan (engl. Spanning tree protocol) pois käytöstä. Muut sillan asetukset (bridge_fd, bridge_hello, bridge_maxage) liittyvät STP-protokollan käyttöön ja ovat suositusten mukaan määritelty varmuuden vuoksi jotta siltaa luotaessa oletusasetukset eivät jäisi voimaan. Esimerkiksi asetus bridge_fd 9 määrittelee sen kuinka kauan silta kuuntelee ja selvittää siihen kytketyn verkon topologiaa ennen liikennöintiä. (Linux Foundation 2009, Li et al. 2009)

4.4 Virtuaalikoneen määrittely ja käyttöönotto

Uusi virtuaalikone voidaan määritellä libvirt – paketin käyttämässä XML – formaatissa, joka kuvaa virtuaalikoneen rakenteen:

```
1       <domain type='kvm'>
2         <name>srv90-01</name>
3         <uuid>708a31ad-223a-994d-692a-8b24fe900100</uuid>
4         <memory>25165824</memory>
5         <currentMemory>25165824</currentMemory>
6         <vcpu>4</vcpu>
7         <os>
8           <type arch='x86_64' machine='pc'>hvm</type>
9           <boot dev='hd'/>
10        </os>
11        <features>
12          <acpi/>
13          <apic/>
14        </features>
15        <clock offset='utc'/>
16        <on_poweroff>destroy</on_poweroff>
17        <on_reboot>restart</on_reboot>
18        <on_crash>destroy</on_crash>
19        <devices>
20          <emulator>/usr/bin/kvm</emulator>
21          <disk type='file' device='disk'>
22            <source file='/srv/vm/srv90-01/srv90-01-system.img'/>
23            <target dev='vda' bus='virtio'/>
24          </disk>
25          <interface type='bridge'>
26            <mac address='00:1f:29:90:01:01'/>
27            <source bridge = 'br-vlan630'/>
28            <model type='e1000' />
29          </interface>
30          <interface type='bridge'>
31            <mac address='00:1f:29:90:01:02'/>
32            <source bridge = 'br-vlan630'/>
33            <model type='virtio' />
34          </interface>
35          <input type='mouse' bus='ps2'/>
36          <graphics type='vnc' port='-1' listen='127.0.0.1' />
37        </devices>
38      </domain>
```

Määritelty virtuaalikone srv90-01 on asetettu käyttämään KVM – pohjaista virtualisointia, joka tarkoittaa että virtuaalikone hyödyntää laiteistopohjaista virtualisointia (HVM) ja siten vaatii virtualisointialustan prosessoreilta tuon sille. (Qemu 2016)

Virtuaalikoneelle on määritelty neljä kappaletta 64 – bittisiä virtuaaliprosessoreja, jotka ovat tyypiltään x86_64 (katso liite A). Tämä yleiseen käyttöön sopiva virtuaaliprosessorityyppi mahdollistaa virtuaalikoneiden migraatiot hyvinkin erilaisten alustakoneiden välillä. Haittana on edistyneimpien CPU – ominaisuuksien puute (kuten SSE4), joka laskee suorituskykyä erityistapauksissa. (Libvirt 2016)

Koneelle on määritelty keskusmuistia 24GB. Memory Balloon driver ei ole käytössä.

Features – haara määrittelee virtuaalikoneen käyttämään ACPI (engl. Advanced Configuration and Power Interface) – standardia, joka määrittelee rajapinnan käytettävän laitteiston tunnistusta ja virranhallintaa varten. Libvirt sisältää toiminnot, jolla virtuaalikone voidaan sammuttaa ACPI:n kautta ohjelmallisesti. Lisäksi haaraan on määri-

telty APIC (engl. Advanced Programmable Interrupt Controller) – laite, jota käytetään virtuaalikoneen raudan keskeytysten hallinnointiin. (Libvirt 2016)

Virtuaalikoneella on kaksi siltaan br-vlan630 – kytkettyä verkkokorttia, toinen on emuloitu Intel E1000 ja toinen paravirtualisointu VirtIO - verkkokortti. Näille on määritelty erikseen käytettävät MAC – osoitteet. Mikäli osoitteita ei määriteltäisi, libvirt generoisi laitteille satunnaiset osoitteet. (Ashley et al. 2016)

Massamuistina toimii paravirtualisoitu tallennuslaite, jonka levytila saadaan virtualisointialustalla olevasta tiedostosta srv90-01-system.img. (Qemu 2016)

XML – tiedoston kuvaama virtuaalikone voidaan määritellä käyttöön Libvirt – pakettiin kuuluvan komennon virsh kanssa. Käsky *virsh define srv90-01.xml* tarkistaa XML – tiedoston ja tallentaa sen tiedot Libvirtin tietokantaan. Virtuaalikone voidaan käynnistää komennolla *virsh start srv90-01*, jolloin KVM – moduuli vastaanottaa lähetetyn ioctl – kutsun ja käynnistää virtuaalikoneen prosessoriprosessien ajamisen kappaleessa 3.2 kuvatulla tavalla.

Edellä kuvattuun virtuaalikoneeseen voidaan sen käynnistämisen jälkeen muodostaa etäyhteys esimerkiksi käyttämällä virt-viewer ohjelmistolla. Tämä onnistuu komennolla:

virt-viewer --connect qemu+ssh://<käyttäjätunnus>
@<virtualisointipalvelin>: <portti>/system srv90-01

Yhteys muodostetaan tässä tapauksessa SSH-tunnelin yli. Haluttaessa voitaisiin käyttää myös salaamatonta yhteyttä komennolla:

virt-viewer --connect qemu://<käyttäjätunnus>@
<virtualisointipalvelin>:<portti>/system srv90-01

Salaamatonta yhteyttä ei tietoturvasyistä kuitenkaan tulisi käyttää kuin luotetusta lähiverkosta.

Virtuaalikoneen toimintaa voidaan virsh – komennolla säädellä erittäin monipuolisesti. Alla olevaan taulukkoon on kuvattu yleisimpien komentojen vaikutukset:

Virsh <komento>	**Kuvaus**
connect	Muodostaa yhteyden virtualisointiytimeen
create	Luo uuden virtuaalikoneen XML-tiedostosta
define	Määrittelee XML-tiedoston mukaisen koneen, mutta ei käynnistä sitä.
desc	Näyttää virtuaalikoneen kuvauksen
destroy	Tuhoaa virtuaalikoneen prosessin, vastaa sähköjen sammuttamista oikeasta koneesta
dominfo	Statistiikka virtuaalikoneesta (domain)
dommemstat	Statistiikkaa virtuaalikoneen muistinkäytöstä
dumpxml	Tulostaa virtuaalikoneen XML – määrittelyn stdouttiin

edit	Edit – komennolla voidaan muokata käyttöön määritellyn virtuaalikoneen XML – tiedostoa
migrate	Siirtää virtuaalikoneen toiselle alustalle
reboot	Uudelleenkäynnistää virtuaalikoneen prosessorit
reset	Resetoi virtuaalikoneen
restore	Palauttaa (tallennetun) virtuaalikoneen ajettavaksi
resume	Jatkaa virtuaalikoneen suorittamista suspendin jälkeen
save	Tallentaa virtuaalikoneen tilan tiedostoon
shutdown	Pyytää virtuaalikoneen virranhallintalaitetta sammuttamaan koneen
start	Käynnistää virtuaalikoneen suorittamisen
suspend	Pysäyttää virtuaalikoneen ajamisen
sysinfo	Virtualisointialustan speksit
undefine	Poistaa virtuaalikoneen määrittelyt Libvirtin tiedoista
vcpuinfo	Näyttää tiedot virtuaalikoneen prosessoreista

Virtuaalikoneen määrittely, käyttöönotto ja operointi komentoriviltä on Libvirt – paketin tarjoamilla työkaluilla suoraviivaista ja nopeaa. Moni korkeamman tason käyttöliittymä nojaakin joko virsh – työkalun tarjoamiin rajapintoihin tai suoraan Libvirt – paketin C – kirjastojen tarjoamiin palveluihin. (Ashley et al. 2016)

5 MITTAUSTULOKSET

Virtualisoinnin yhtenä kompastuskivenä on yleensä pidetty negatiivista vaikutusta palvelimien suorituskykyyn. Puhtaasti ohjelmistopohjaisen virtualisoinnin tapauksessa näin toki olikin, mutta nykyaikainen laitteistoavusteinen virtualisointi on korjannut lukuisia ongelmia suorituskyvyn saralla. Kuten toisessa luvussa todettiin, tarjoaa laitteistoavusteinen virtualisointi lähes natiivia suorituskykyä erityisesti prosessorien virtualisoinnin kanssa. (Huynh et al. 2013, Kourai et al. 2014)

Tässä luvussa käydään mittaustulosten avulla läpi eri palvelinvirtualisoinnin osa-alueilla tapahtunut kehitys. Mittauksissa on rajoituttu best practice – ratkaisujen suosittelemiin vaihtoehtoihin (IBM 2012).

Mittauslaitteistossa on käytettävissä Intelin i5-2520M – prosessori, joka sisältää kaksi prosessoriydintä. Kumpikin prosessoriytimistä käyttää Intelin kehittämää hyperthreading – tekniikkaa (HT), jossa useaa ohjelmasäiettä voidaan ajaa prosessorissa näennäisesti samanaikaisesti. Tekniikan tuoma nopeusetu perustuu prosessorin eri osien hyötysuhteen parantamiseen esimerkiksi ohjelmakoodin suorittamisen keskeyttävien välimuistihakujen aikana. Nopeusetu on tyypillisissä palvelinohjelmistoissa noin 30% luokkaa. (Marr et al. 2002)

5.1 Prosessorin suorituskyky

Prosessorin suorituskyvystä, tai pikemminkin virtualisoinnin suorituskyvystä, kertoo mittaustesti, jossa virtuaaliprosessorin suorituskykyä verrataan alustana toimivan koneen prosessorin suorituskykyyn.

Sysbench – testissä prosessorit laskevat kaikki alle 20000 kokoiset alkuluvut. Testi ajettiin 1 – 4 samanaikaisella Sysbench - laskentasäikeellä. Alustakoneella oli käytettävissä neljä loogista prosessoriydintä, eli kaksi fyysista ydintä sekä kaksi HT – tekniikan käytöstä johtuvaa säiettä. Virtuaalikoneille oli määriteltynä yksi, kaksi tai neljä loogista prosessoriydintä. Jälkimmäisen vaihtoehdon tarkoituksena oli peilata mahdollisimmat tarkasti alustakoneen prosessorikokoonpanoa.

Testin tarkoituksena oli selvittää, miten virtuaalikoneiden prosessorin suorituskyky on verrattavissa alustakoneen prosessorin suorituskykyyn, ja kuinka radikaalisti tulokset muuttuvat kun ajettavien säikeiden määrä on prosessoriytimen määrää suurempi. Nopeustesti suoritettiin kymmenen kertaa ja tuloksista laskettiin keskiarvo. Virtuaalikoneiden suorituskykyä on myös verrattu suhteessa virtualisointialustan samaan laskentaan käyttämään aikaan. Tulokset on esitelty kuvassa 5.1.

Testi ajettiin komennolla:
sysbench --test=cpu --cpu-max-prime=20000 --num-threads=n run, jossa n on ajettavien laskentasäikeiden määrä.

	Sysbench – säikeet			
	1	2	3	4
Virtualisointialusta				
Intel Core i5-2520M	26,9047 s	15,5624 s	10,5183 s	8,2982 s
2 CPU + 2 HT	1	1	1	1
Virtuaalikone				
Qemu Virtual CPU version 2.0.0 64bit	26,9957 s	27,0073 s	27,0247 s	27,0769 s
1 CPU + 0 HT	1,0034	1,7354	2,5693	3,2630
Virtuaalikone				
Qemu Virtual CPU version 2.0.0 64bit	27,0134 s	15,5997 s	14,3883 s	14,3944 s
2 CPU + 0 HT	1,0040	1,0024	1,3679	1,7346
Virtuaalikone				
Qemu Virtual CPU version 2.0.0 64bit	27,0075 s	15,5801 s	10,5542 s	8,3325 s
2 CPU + 2 HT	1,0038	1,0011	1,0034	1,0041

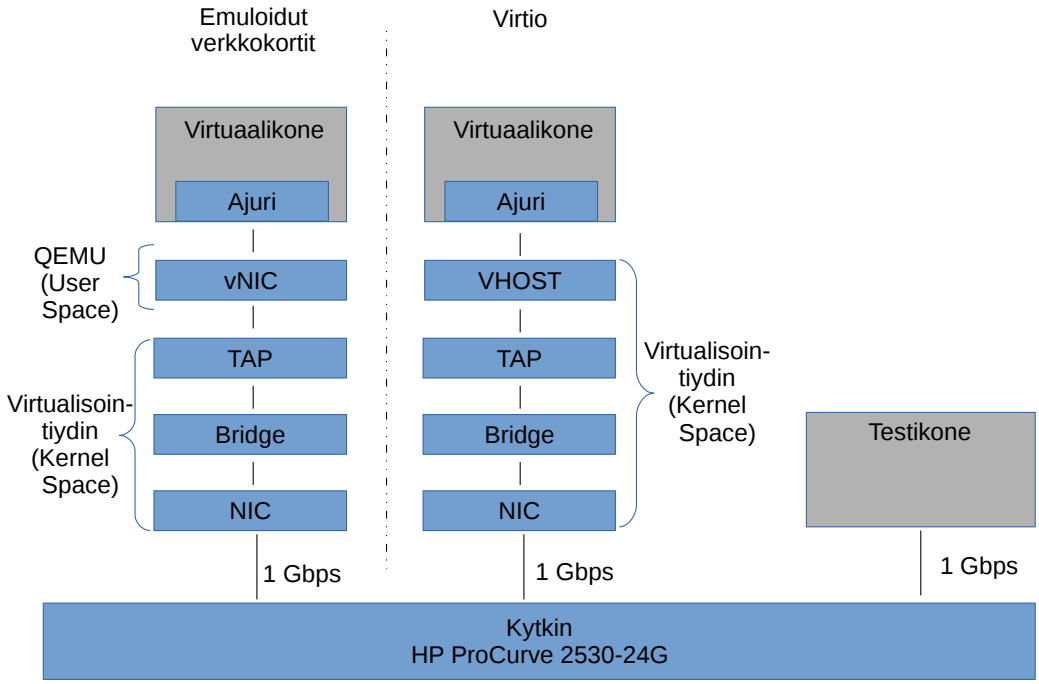
Kuva 5.1: Sysbench-testiä käytettiin selvittämään suorituskykyeroa virtualisoidun prosessorin ja alustan prosessorin välillä, ja eri konfiguraatioiden vaikutusta tuloksiin. Pienempi aika parempi. Ajan alla myös suhdeluku alustan aikaan verrattuna.

Mittaustulokset yllättävät johdonmukaisuudellaan: Virtuaalikoneiden Qemu64 – prosessorien suorituskyky on samaa luokkaa varsinaisen alustaprosessorin kanssa, kun huomioidaan virtuaalikoneissa oleva virtuaaliprosessorien määrä. Laskentasäikeiden määrän ylittäessä virtuaalikoneiden käytettävissä olevien loogisten prosessoriytimien määrän mittaustulokset pysyvät lähestulkoon vakiona. Tämä on erityisen hyvin havaittavissa 1 CPU + 0 HT – ytimen virtuaalikoneessa: koska testissä lasketaan kaikki alkuluvut 20000 asti, monen säikeen ajaminen tämän virtuaalikoneen yhdellä prosessoriytimellä ei ole sen nopeampaa kuin saman laskusuorituksen ajaminen yhdessä säikeessä. Sama ilmiö toistuu 2 CPU + 0 HT – ytimen virtuaalikoneessa kahden laskentasäikeen kohdalla. Toisaalta, koska alkulukujen laskentatehtävä on helposti säikeistettävissä, nähdään sekä virtualisointialustan että 2 CPU + 2 HT – ytimen virtuaalikoneen tuloksista laskenta-ajan laskevan melko suoraan suhteessa prosessoriytimien määrään. Kolmella ja neljällä laskentasäikeellä näiden mittaustulokset paranevat hieman hitaammin, juuri kuten Hyperthreading – tekniikkaa käyttäessä pitäisikin käydä. Mittaustulokset ovat molemmissa tapauksissa kuitenkin hyvin samaa tasoa.

Näiden testitulosten perusteella näyttäisi siltä, että laitteistoavusteinen virtualisointi ei oleellisesti hidasta prosessorin toimintaa. Tuloksia verrataan myöhemmässä aliluvussa muiden julkaisujen tuloksiin.

5.2 Verkkoliikenteen suorituskyky

Verkkoliikenteen suorituskykyä KVM – virtualisointialustassa mitattiin vertailemalla paravirtualisoitua verkkolaitetta (VirtIO), kahta emuloitua verkkolaitetta (Intel E1000- ja Realtek RTL8139 – piireihin perustuvia) ja virtualisointialustan Intel 82579LM – verkkolaitetta keskenään. Vertailua varten ajettiin kaksi erityyppistä mittausta, joissa molemmissa mittaukset toistettiin viisi kertaa sekä palvelimelta lähtevään että sille saapuvaan verkkoliikenteeseen kohdistuen. Tuloksista laskettiin keskiarvo ja keskihajonta analysoinnin helpottamiseksi. Testeissä käytetty virtualisointialusta sekä testin suorittava kone oli kytketty toisiinsa HP ProCurve 2530-24G – kytkimen välityksellä yhden gigabitin linkillä. Virtuaalikoneiden verkkolaitteet on kytketty virtualisointialustan verkkosillan kautta koneen fyysiseen verkkolaitteeseen luvussa 3.4 kuvatun tavan mukaisesti. Testikokoonpano on esitetty myös kuvassa 5.2.



Kuva 5.2: Verkkomittauksissa käytetty testikokoonpano. Kuvassa eritelty emuloidun ja paravirtualisoidun verkkoarkkitehtuurin erot. Virtuaalikoneet ja testikone on kytketty HP:n kytkimen kautta toisiinsa yhden gigabitin verkkolinkillä. Kuva mukaillen (Li et al. 2009) ja (Burns 2010).

Ensimmäisen mittauksen tulokset on esitelty kuvassa 5.3. Se mittaa verkkolaitteiden maksimaalista suorituskykyä iperf – ohjelmiston avulla: iperf lähettää verkkolinkin yli dataa testattavien koneiden keskusmuistista toiseen ja raportoi tähän käytetyn ajan. Iperfin syötteenä voidaan haluttaessa käyttää myös mitä tahansa stdin – syötettä tai tiedostoa, esimerkiksi /dev/urandom (Gates et al. 2003). Mikäli tulosten halutaan kuvastavan verkkolinkin siirtonopeutta, syötteenä käytettävän datalähteen tulisi olla sitä nopeampi.

Mittaus on tehty TCP – protokollaa käyttäen. TCP:n vuonohjaus on molemmissa mittauksissa jätetty käyttöjärjestelmän vastuulle, joka säätää ikkunan kokoa tarpeen mukaan. Näin saadaan mahdollisimman käytännönläheiset mittaustulokset. (Stallings 2012)

Mittausohjelmisto käynnistettiin testipalvelimella komennolla *iperf -s -t*. Tämä määrittely asettaa ohjelmiston kuuntelemaan oletusporttia TCP 5001. Testaavalla laitteella iperf ajettiin komennolla *iperf -c IP-OSOITE -r*. Vipu -r ajaa testin molempiin suuntiin peräjälkeen. (Gates et al. 2003)

	Mittaukset [Mbit/s]					Keski-arvo [Mbit/s]	Keski-hajonta [Mbit/s]
	1.	2.	3.	4.	5.		
Virtualisointialusta (Intel 82579LM)							
output	936	936	935	936	936	935,8	0,4
input	933	931	926	928	926	928,8	3,1
Virtuaalikone (E1000)							
output	933	933	933	931	932	932,4	0,9
input	927	918	916	917	915	918,6	4,8
Virtuaalikone (VirtIO)							
output	935	935	935	935	935	935	0,0
input	933	934	933	933	933	933,2	0,4
Virtuaalikone (RTL8139)							
output	417	406	395	412	395	405	9,9
input	292	309	308	324	314	309,4	11,6

Kuva 5.3: Verkon nopeustestit iperf - ohjelmistolla. Mitattuna liikenne palvelimelta verkkoon (output) ja verkosta palvelimelle (input).

Iperf – ohjelmalla päästään hyvin lähelle verkkolinkin teoreettista maksiminopeutta. Mittaustuloksista nähdään, että virtualisointialustan fyysinen verkkokortti, paravirtualisoitu VirtIO sekä emuloitu E1000 olivat hyvin lähellä tätä rajaa niin vastaanottokuin lähetysnopeuksissakin. Sen sijaan emuloidun RTL8139 – piirin suorituskyky jäi erittäin kauas tästä. Kaikissa mittauksissa keskihajonta oli melko pieni, toisin sanoen mittaustulokset olivat hyvin johdonmukaisesti samaa suuruusluokkaa.

Toinen mittaus, jonka tulokset on esitelty kuvassa 5.4, mittasi 1 gigatavun tiedoston siirron verkon yli laitteelta toiselle. Tämä suoritettiin käyttämällä ohjelmia dd ja nc (netcat). Testaavalla laitteella generoitiin 1 gigatavu dataa, joka putkitettiin yhden megatavun palasina dd:n stdoutin kautta nc – ohjelman stdin – syötteeksi, ja edelleen verkkolinkin yli testipalvelimella käynnissä olevalle nc – ohjelmalle, joka kuuntelee TCP – porttia 12345. Tämä toteutettiin komennolla *dd if=/dev/zero bs=1M count=1K | nc -vvn IP-OSOITE 12345*.

Testipalvelimella ajettiin komento *nc -vvlnp 12345 >/dev/null*. Tämä määrittelee nc – ohjelman kuuntelemaan TCP – porttia 12345 ja kieltää sitä tekemästä DNS – nimenselvitystä yhteyden aikana. Ohjelman verkon kautta saama syöte ohjataan laitteelle /dev/null toisin sanoen dataa ei tallenneta mihinkään. Jos data tallennettaisiin massamuistille, vaikuttaisi tämä mittaustuloksiin mikäli massamuisti on verkkolinkkiä hitaampi.

	Mittaukset [MB/s]					Keski-arvo [MB/s]	Keski-hajonta [MB/s]	Keski-arvo [Mbit/s]
	1.	2.	3.	4.	5.			
Virtualisointialusta (Intel 82579LM)								
output	117	117	117,1	117	117	117,02	0,0	936
input	117	117	117	117	117	117	0,0	936
Virtuaalikone (E1000)								
output	117	117	117	117	112	116	2,2	928
input	29,8	31,6	66,1	82,2	44,9	50,92	22,7	407
Virtuaalikone (VirtIO)								
output	117	117	117	117	117	117	0,0	936
input	117	116,9	117	117	117	116,98	0,0	936
Virtuaalikone (RTL8139)								
output	77,3	63,5	59,5	74,7	59,3	66,86	8,6	535
input	21,6	18,8	19,9	18	27,7	21,2	3,5	170

Kuva 5.4: Verkon nopeustestit netcat- ja dd - ohjelmistolla. Mitattuna liikenne palvelimelta verkkoon (output) ja verkosta palvelimelle (input).

Tuloksista havaitaan, että paravirtualisoitu VirtIO on jälleen käytännössä yhtä nopea virtualisointialustan verkkokortin kanssa. Molemmat ovat hyvin lähellä linkin teoreettista maksiminopeutta. E1000:n palvelimelle saapuva kaista on kuitenkin merkittävästi hitaampi kuin edellisissä mittauksissa, ja mittausarvojen välillä on hajontaa melkoisesti. Niin ikään myös emuloidun RTL8139:n saapuva kaista on melko hidas, joskin hajontaa sen mittaustuloksissa ei juuri ole.

Yhteenvetona verkon suorituskykymittauksista voitaneen todeta, että VirtIO – ajuria käytettäessä suorituskyky on käytännössä natiivilla tasolla. Intelin E1000 – emulaa-

tio toimii myös kohtalaisen hyvin. Realtekin RTL8139:ä ei tuotantokäyttöön juurikaan voi suositella.

5.3 Levyohjaimien suorituskyky

Testeissä käytettiin virtualisointialustan fyysisen levyohjaimen lisäksi paravirtualisoitua VirtIO – levyohjainta, sekä emuloituja IDE- ja SCSI – ohjaimia. Niillä ohjattiin virtuaalisia massamuisteja, jotka saivat tallennuskapasiteettinsa virtualisointialustan tiedostojärjestelmässä sijaitsevasta RAW – levykuvatiedostosta best practice – menetelmien mukaisesti.

Ensimmäinen, Sysbench – ohjelmalla ajettu testi mittaa levyjärjestelmän suorituskykyä tiedostoja kirjoitettaessa ja lukiessa. Sitä varten on luotu valmiiksi 60GB tiedostoja komennolla *sysbench --test=fileio –file-total-size=60G prepare*. Varsinainen testi voidaan ajaa tämän jälkeen komennolla *sysbench --test=fileio –file-total-size=60G --file-test-mode=rndrw --init-rng=on --max-time=300 --max-requests=0 run*. Asetus rndrw tarkoittaa yhdistettyä kirjoitus- ja lukutestiä, jossa levyoperaatioita suoritetaan satunnaisessa järjestyksessä. Testi kestää 300s, eikä sen aikana rajoiteta i/o – pyyntöjen määrää. Tulokset on esitelty alla olevassa kuvassa numero 5.5. (Kopytov 2009)

	Mittaukset	Nopeus suhteessa virtualisointialustaan
Fyysinen kone Virtualisointialusta	11,740 Mb/s	100,0%
Virtuaalikone (emuloitu IDE – ohjain)	5,829 Mb/s	49,6%
Virtuaalikone (Paravirtualisoitu VirtIO-ohjain)	7,833 Mb/s	66,7%
Virtuaalikone (emuloitu SCSI – ohjain)	6,630 Mb/s	56,5%

Kuva 5.5: Sysbenchin fileio - testin tulokset. Tulokset mittaavat levyjärjestelmän suorituskykyä erikokoisilla tiedostoilla.

Kuten kuvasta käy ilmi, paravirtualisoitu VirtIO – levyohjain on lähinnä fyysisen koneen levyohjaimen nopeutta. Emuloitujen ohjaimien välillä ei ole valtaisaa suorituskykyeroa.

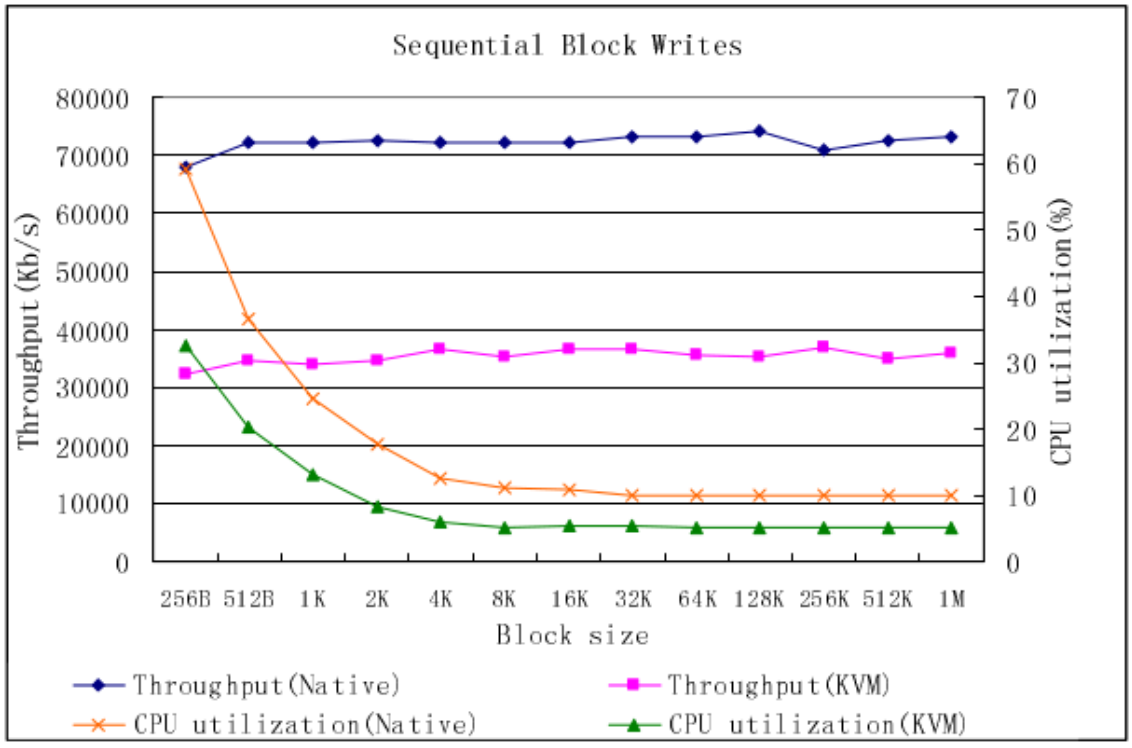
Toinen testi puolestaan mittaa levyjärjestelmän suorituskykyä kirjoitettaessa yhtä suurta, tässä tapauksessa kymmenen gigatavun, tiedostoa. Mittaukset on tehty dd – ohjelman avulla komennolla *dd if=/dev/zero of=mittaus.img bs=10MB count=1K*, joka kirjoittaa levylle mittaus.img – tiedoston kymmenen megatavun lohkoissa. Tulokset on esitelty kuvassa 5.6.

	Mittaukset	Nopeus suhteessa virtualisointialustaan
Fyysinen kone Virtualisointialusta	372 MB/s	100,0%
Virtuaalikone (emuloitu IDE – ohjain)	271 MB/s	72,8%
Virtuaalikone (Paravirtualisoitu VirtIO-ohjain)	359 MB/s	96,5%
Virtuaalikone (emuloitu SCSI – ohjain)	299 MB/s	80,4%

Kuva 5.6: Levyn kirjoitusnopeus testattuna dd - ohjelmalla, yksi 10GB tiedosto.

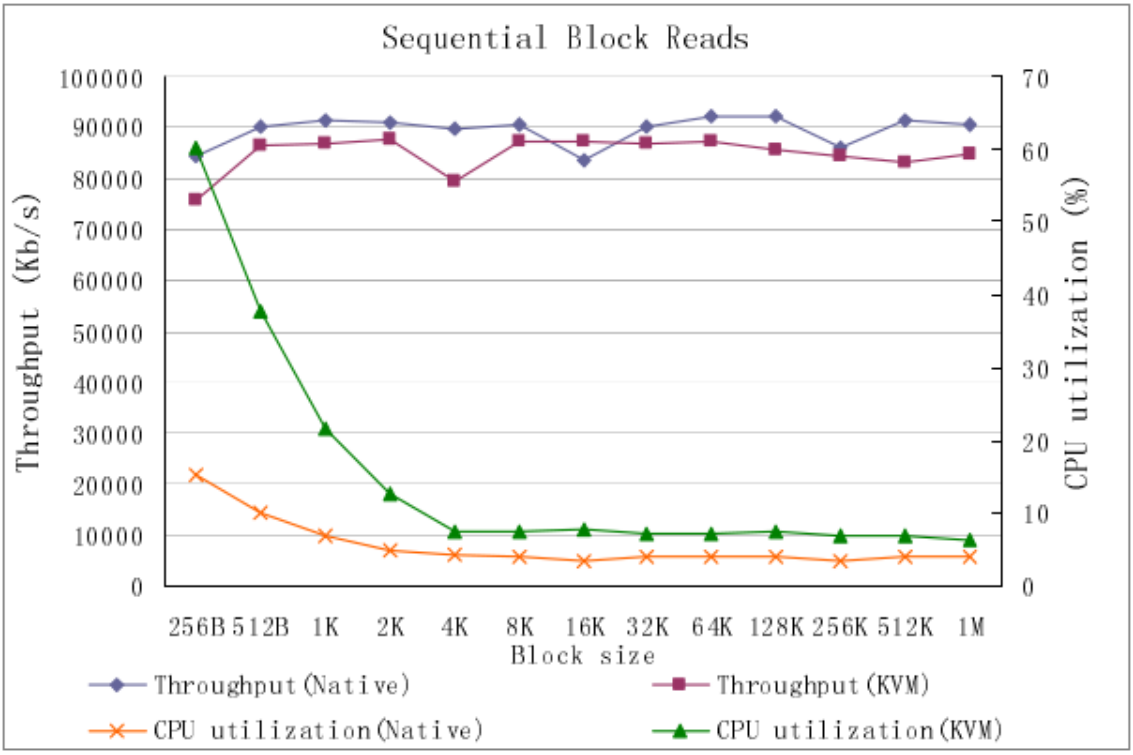
Yhdellä isolla tiedostolla testattaessa VirtIO:n sekä emuloitujen levyohjaimien suorituskyky on lähempänä natiivia kuin aiemmassa satunnaisia kirjoitus- ja lukuoperaatioita mittaavassa Sysbench – testissä. Emuloitujen ohjaimien välinen suorituskykyero ei ole merkittävä.

Levyohjaimien testit viittaisivatkin siihen suuntaan, että virtualisointi hidastaa etenkin paravirtualisoitua levyohjainta käytettäessä yksittäistä kirjoitusoperaatiota suhteellisen vähän, mutta lukuisten operaatioiden kohdalla hidastuksen vaikutus kasvaa. Julkaisussa (Zhang et al. 2010) – on suoritettu joukko mittauksia, jotka valaisevat syitä tähän. Tulokset on esitelty kuvissa 5.7 ja 5.8:



Kuva 5.7: Kirjoitusnopeustesti, virtuaalikoneen ja virtualisointialustan suorituskyky ja CPU – kuorma kirjoitettavan lohkon koon funktiona. (Zhang et al. 2010)

Kuvasta nähdään, että virtuaalikoneen kirjoitusnopeus on noin puolet alustakoneen kirjoitusnopeudesta. Virtuaalikoneen prosessorin käyttöaste on kuitenkin alustakonetta suurempi etenkin käytettäessä pienempiä 4K ja alle lohkokokoja. (Zhang et al. 2010)



Kuva 5.8: Lukunopeustesti, virtuaalikoneen ja virtualisointialustan suorituskyky ja CPU – kuorma luettavan lohkon koon funktiona.(Zhang et al. 2010)

Vastaavan tyyppinen ilmiö nähdään myös lukutestissä. Virtuaalikoneen suorituskyky on hyvin samaa luokkaa alustakoneen kanssa, mutta lukuoperaatioiden aiheuttama prosessorikuorma on merkittävästi suurempi etenkin käytettäessä pienempiä lohkokokoja. Syynä tähän on se, että pienempi lohkokoko parantaa todennäköisyyttä sille, että seuraava haettava data löytyy levyn välimuistista, eikä hidasta lukua levyn pinnalta tarvitse tehdä. Pieni lohkokoko aiheuttaa kuitenkin enemmän käsiteltävää i/o – liikennettä virtualisointiytimelle. Tämä puolestaan aiheuttaa enemmän overheadia varsinaiseen luettavan datan määrään nähden. (Zhang et al. 2010)

Aiemmat Sysbench – testin tulokset vastannevatkin paremmin palvelimien levyi/o:n todellisuutta kuin dd:llä suoritettu testaus. Näin ollen voidaankin johtopäätöksenä todeta, että tallennustilan virtualisoinnissa ei vielä olla niin lähellä natiivia suorituskykyä kuin prosessorin ja tietoliikenteen virtualisoinnissa ollaan. Kuten aliluvuissa 2.3 ja 3.5 on aiemmin todettu, tallennuskapasiteetin virtualisoinnin myötä palvelimien levytila on kuitenkin yhä useammin osoitettu SAN- tai NAS – tallennusjärjestelmiltä esimerkik-

si iSCSI – tekniikan avulla. Tällöin levyohjainten virtualisoinnin suorituskyvyn tärkeys vähenee ja tietoliikenteen virtualisoinnin vaikutus kasvaa, kun tarkastellaan palvelimien tallennusjärjestelmien kehityssuuntaa kokonaisuutena.

5.4 Mittausten vertailu muiden julkaisemiin tuloksiin

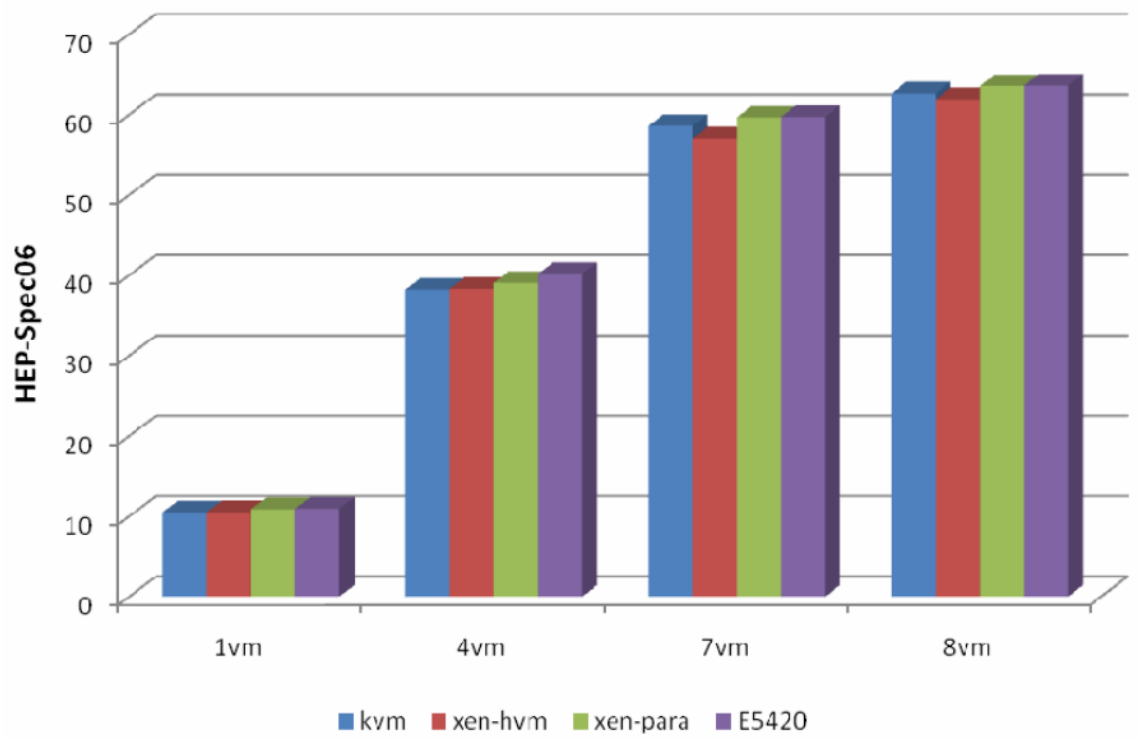
Tässä aliluvussa verrataan omia mittaustuloksia muiden julkaisemiin mittaustuloksiin, ja analysoidaan havaintoja. Julkaisuihin viitataan esittelyn jälkeen tekstissä lähdeviittauksella.

Julkaisussa (Chierici et al. 2010) ”A quantitative comparison between xen and kvm” vuodelta 2010 kirjoittajat vertailevat Xeniä ja KVM:ää toisiinsa sekä alustana toimivan koneen suorituskykyyn. Julkaisu on hieman vanha, joten suoria vertailuja omien ja julkaisujen mittaustulosten välillä ei voida tehdä. Julkaisu kuitenkin sisältää mielenkiintoisia CPU- ja verkkonopeustestejä, joista kaksi esitellään myöhemmin.

Toinen verkkonopeustesteissä käsiteltävä julkaisu on (Zhang et al. 2010) ”Evaluating and Optimizing I/O Virtualization in Kernel-based Virtual Machine (KVM)”, joka sekin on vuodelta 2010. Julkaisussa evaluoidaan menetelmiä, joilla on vaikutusta KVM:n I/O – suorituskykyyn.

Julkaisu (Soriga et al. 2013) ” A comparison of the performance and scalability of Xen and KVM hypervisors” vuodelta 2013 vertaa sekin Xeniä ja KVM:ää keskenään. Julkaisussa on vertailtu tarkasti näiden levy-I/O – suorituskykyä verrattuna virtualisointialustan nopeuteen. Samaan teemaan liittyen, julkaisussa (Kourai et al. 2014) ”Efficient VM Introspection in KVM and Performance Comparison with Xen” vuodelta 2014 on erittäin hyvin vertailtu levykuvien RAW ja qcow2 välisiä eroja KVM- ja XEN – alustoilla.

Julkaisussa (Chierici et al. 2010) on tutkittu Intel Xeon E5420 – prosessorilla, jossa on neljä fyysistä prosessoriydintä, yhden virtuaaliprosessorin virtuaalikoneiden ajoa eri määrinä testissä mukana olevilla alustoilla. Testissä virtuaalikoneita kuormitettiin HEP-Spec06 – testiohjelmalla. Tulokset on esitetty kuvassa 5.9.

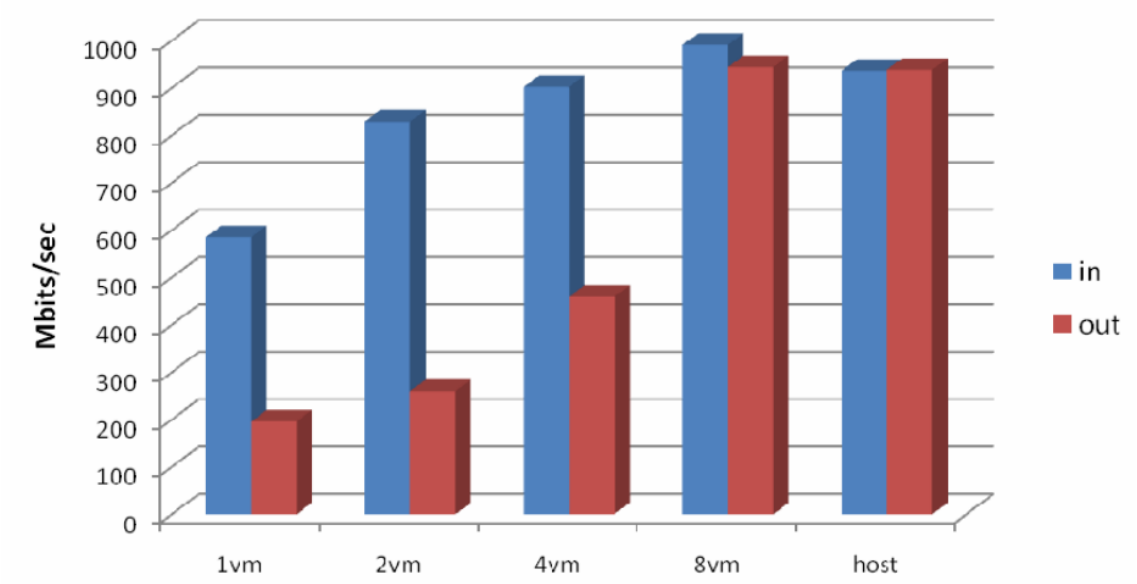


Kuva 5.9: HEP-Spec06 - testiohjelman ajoa eri määrässä virtuaalikoneita ja verrattuna prosessorin natiiviin suorituskykyyn. Xeon E5420 – prosessorissa on neljä fyysistä prosessoriydintä, jotka tukevat hyperthreading – tekniikkaa. Tämä selittää eron suorituskyvyssä neljää useammalla virtuaalikoneella. (Chierici et al. 2010)

Julkaisussa havaittiin eri virtualisointiohjelmat hyvin tasaväkisiksi kyseisessä testissä. Niiden nopeus oli myös hyvin lähellä natiivia suorituskykyä. Lisäksi osoittautui, että testialustan neljän ytimen prosessorilla voitiin ajaa jopa seitsemää tai kahdeksaa testiohjelmaa pyörittävää virtuaalikonetta samanaikaisesti ilman että suorituskyky merkittävästi heikkenee.

Tämä johtuu samasta ilmiöstä, joka vaikutti omassa mittauksessa (kuva 5.1). Kuten luvun alussa mainittiin, Intelin prosessoreissaan käyttämä hyper-threading – tekniikka parantaa yksittäisen prosessoriytimen suorituskykyä noin 30 prosentin luokkaa (Marr et al. 2002). Tämä selittää kuvan 5.9 tulokset hyvin, ja on yhteneväinen omien mittaushavaintojen kanssa.

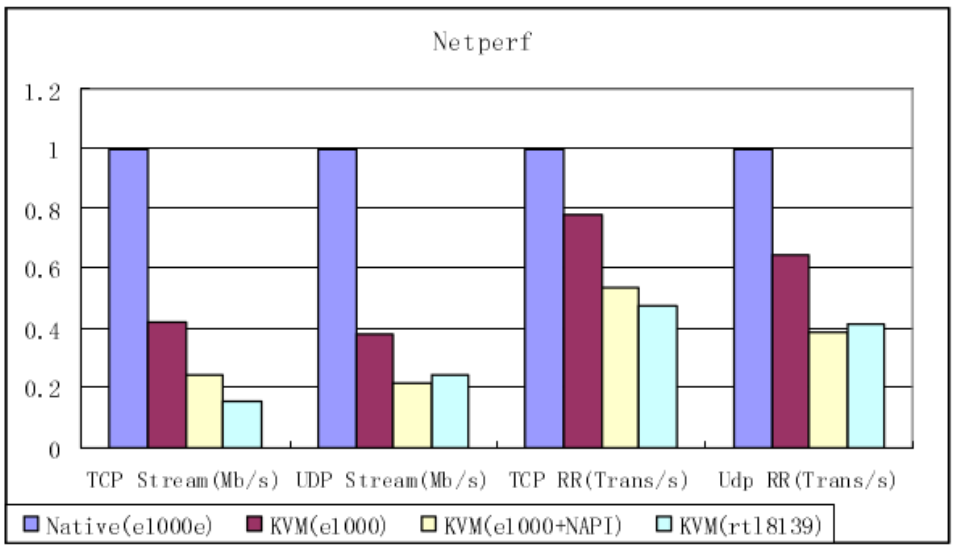
Samassa julkaisussa käsiteltiin myös verkkomittauksia. Samalla alustalla ajettiin yhdestä kahdeksaan virtuaalikonetta, joiden verkkokorttina oli emuloitu intel E1000. Alustakoneessa oli 1 gigabitin linkki. Mittaukset on tehty käyttäen iperf – ohjelmaa.



Kuva 5.10: Emuloidun E1000 – verkkokortin nopeusvertailua virtualisointialustaan nähden iperf – ohjelmalla. Vasta kahdeksan testiä ajavaa virtuaalikonetta saivat fyysisen linkin täyteen. (Chierici et al. 2010)

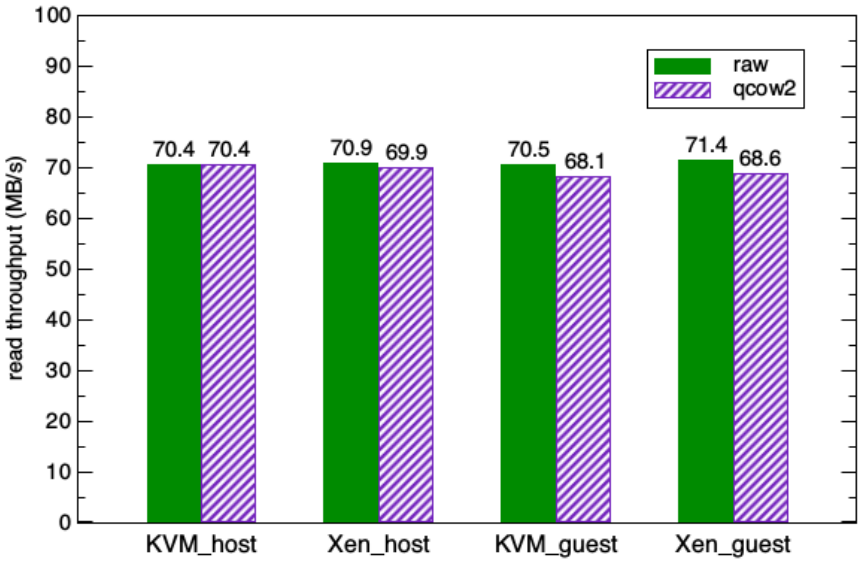
Kuvasta 5.10 nähdään, että alustakoneella saavutetaan lähes teoreettinen gigabitin linkin maksiminopeus, samoin kuten omissa mittauksissa kuvissa 5.3 ja 5.4. Yhdellä virtuaalikoneella ei kuitenkaan saavutettu yhtä hyviä mittaustuloksia kuin omassa kuvassa 5.3, mutta ilmiö oli samansuuntainen kuvan 5.4 tulosten kanssa: Emuloitu E1000 ei yksinkertaisesti kykene täyttämään koko gigabitin linkkiä. Se että (Chierici et al. 2010):n tulokset ovat omiin tuloksiin verrattuna heikompia, selittynee julkaisun iällä. E1000 – kortin emulointi on oletettavasti kuuden vuoden aikana ottanut aimo harppauksia parempaan suuntaan. Julkaisun tuloksista nähdään, että koko linkki saadaan täyteen vasta ajamalla samanaikaisesti kahdeksaa virtuaalikonetta ja iperf – testiä. Julkaisussa epäilläänkin, että E1000 – verkkoajurin ohjelmakoodi olisikin sisältänyt tarkoituksellisia rajoituksia laitteen nopeuteen, jolla saavutettaisiin parempi luotettavuus.

Samalta vuodelta oleva (Zhang et al. 2010) - julkaisu vertailee netperf – ohjelmalla emuloituja verkkokortteja alustakoneen oikeaan E1000e – kortin nopeuksiin. Tulokset antavatkin vahvistusta epäilyille vuoden 2010 E1000 – emuloinnin nykyistä heikommalle suorituskyvylle, molempien julkaisujen mittauksissa E1000 – emuloinnin TCP – streamin nopeus on noin 40% luokkaa fyysisen verkkokortin nopeudesta. Julkaisun mittaustulokset on esitelty kuvassa 5.11.



Kuva 5.11: Netperf - mittaustulokset. Vertailu emuloitujen ja fyysisen verkkokortin nopeuksista. (Zhang et al. 2010)

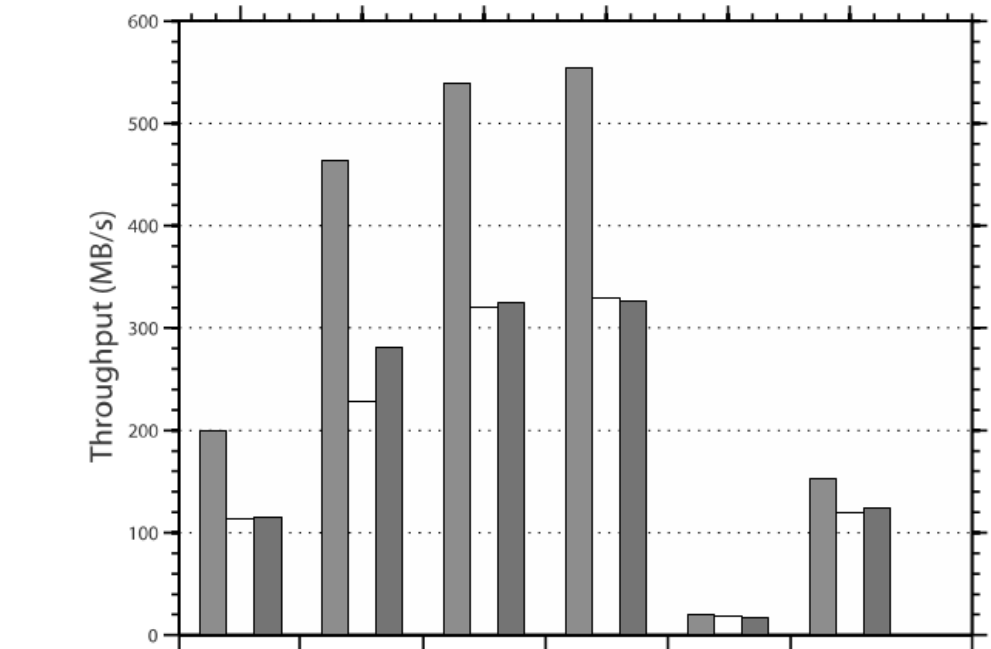
Julkaisussa (Kourai et al. 2014) oli mielenkiintoinen lukunopeusvertailu RAW- ja qcow2 – levykuvaformaattien välillä. Siinä ei ainoastaan verrattu suorituskykyä virtuaalikoneiden kanssa, vaan samat testit tehtiin mappaamalla levykuvat suoraan virtualisointialustan tiedostohierarkiaan. Mittaustulokset olivat lähes identtisiä eri virtualisointijärjestelmien välillä, mutta myös virtuaalikoneiden ja alustakoneiden välillä. Tulokset on esitelty kuvassa 5.12.



Kuva 5.12: Levykuvatiedostoformaattien nopeusvertailu, kyseessä lukunopeustesti (Kourai et al. 2014)

Sinänsä ei ole mikään yllätys, että RAW- ja qcow2 – formaattien lukeminen on yhtä nopeata virtuaalikoneiden kesken, sillä pohjimmiltaan kysymys on täysin identtisestä toiminnosta: molemmissa tapauksissa luetaan valmista dataa levykuvat sisältävältä massamuistilta. Suorituskykyero näkyisi sen sijaan kirjoitustestissä, jota julkaisussa ei valitettavasti ollut mukana. Mittauksen todellinen anti liittyi kuitenkin vertailuun virtuaalikoneen ja alustakoneen nopeuksien välillä. Nämä nopeudet olivat lähes identtisiä. Tämä tarkoittaa siis sitä, että julkaisun mittausten mukaan itse virtualisointiin käytetty prosessoriaika on hyvin mitätöntä. Tuloksia olisi ollut mukava verrata virtualisointialustan levyjärjestelmän natiiviin suorituskykyyn, mutta valitettavasti julkaisussa ei ollut tämäntyyppisia mittaustuloksia.

Julkaisu (Soriga et al. 2013) sisälsi mittaussarjan, jossa verrattiin kattavasti erityyppisiä levy-I/O – aktiviteetteja alustakoneen ja VirtIO – levyohjainta käyttävän virtuaalikoneen välillä. Virtualisointiin oli käytetty KVM:ää. Mittaustulokset on esitelty kuvassa 5.13.



	Write	Rewrite	Read	Re-read	Random read	Random write
Bare Metal	200.53	463.31	539.70	543.37	20.19	152.55
KVM (virtio)	114.10	228.97	320.32	329.14	17.71	119.76
Xen (para)	115.49	281.07	323.99	326.42	16.44	124.19

Kuva 5.13: Levy-I/O - nopeustestien tulokset. KVM:n ja Xenin päällä ajetaan yhtä virtuaalikonetta, vertailun vuoksi virtualisointialustan tulokset. (Soriga et al. 2013)

Mittaustuloksista käy ilmi, että virtuaalikoneen levy-I/O:n suorituskyky on noin 60% luokkaa verrattuna alustakoneeseen. Tämä on hyvin samaa luokkaa kuvassa 5.5 esiteltyjen omien mittausten kanssa. Tulosten yhteneväisyys vihjaa siis, että omien mittausten mittausmetodit ovat valideja.

6 YHTEENVETO

IT-järjestelmien virtualisointi on pitkän 60 – luvun keskustietokoneista alkaneen matkansa aikana muuttunut ja jalostunut merkittävästi ennen saapumistaan nykypäivän palvelimiin ja jopa työasemiin. Tekniikan perusteet ja monet IBM:n tekemät merkittävät ratkaisut vaikuttavat virtualisointiarkkitehtuureihin vielä nykypäivänä. Vaikka osa teoriasta onkin vuosien varrella muuttunut merkityksettömäksi, se miksi virtualisointia tehdään, sekä sen edut ja mahdollisuudet eivät ole kuitenkaan hävinneet.

Uudemmatkin oivallukset, kuten VMWaren Binary Translation – tekniikka ja prosessorivalmistajien x86 – käskykantaan lisäämät virtualisointilaajennukset ovat jättäneet alaan kiistattoman jälkensä. Ne ovat samalla mahdollistaneet aivan uudenlaisia tapoja virtualisoida tietokonejärjestelmiä.

Eräs näiden oivallusten seurauksena syntynyt järjestelmä on tässä työssä käsitelty Kernel-based virtual machine. Se on avoimen lähdekoodin virtualisointialusta, joka koostuu Linux - ytimestä, siihen lisätystä kvm - moduulista sekä laite-emuloinnin tarjoavasta QEMU - ohjelmistopaketista. Sitä on jo pidempään käytetty menestyksekkäästi virtualisoinnissa hyvin laajalla skaalalla aina yksittäisistä työasemista pilvipalveluluiden konesaleihin asti. Ohjelmisto perustuu avoimen lähdekoodin ratkaisuihin, mutta sille on tarjolla tukipalveluita lukuisten kaupallisten tarjoajien toimesta. Pelkästään ohjelmiston suosio ja sen tukipalveluiden parissa tehtävä bisnes on osoitus ratkaisun toimivuudesta ja käyttövalmiudesta. KVM – virtualisointialustaa varten on kehitetty myös paljon uusia menetelmiä ja tekniikoita jotka parantavat ja laajentavat sen toimintaa suuresti. Tämä kehitys tuskin on hidastumassa, sillä avoimen lähdekoodin suurin kiistaton etu, sen vapaa muunneltavuus, on todella vahva eteenpäin vievä voima. Muita etuja on edullinen käyttöönotto sekä tekniikan avoimuuden tuoma turvallisuus ja jatkuvuus. Mikäli KVM:n kehittäjät yhtäaikaisesti päättäisivät lopettaa jatkokehityksen, tuote ja lähdekoodi sekä niihin panostettu työ ei häviä mihinkään. Näin kehitystyötä on helppo jatkaa uusien ihmisten voimin. Nämä edut näkyvät myös KVM:n ympärille rakennetun ekosysteemin kehityksessä. Hyvin rakennetun virtualisointiratkaisun ympärille on ilmestynyt valtava määrä kolmansien osapuolien hallintatyökaluja, uusia ominaisuuksia ja uudella tapaa paketoituja virtualisointiohjelmistoja, jotka hyödyntävät olemassaolevaa tekniikkaa.

KVM perustuu laitteistoavusteiseen virtualisointiin, jonka x86 - arkkitehtuurin prosessorikehitys on mahdollistanut. Koska laitteistot tarjoavat virtualisointia tukevia palveluita, virtualisointiohjelmiston ei tarvitse suorittaa niin paljon emulaatiota kuten aiemmin puhtaasti ohjelmistopohjaiseen virtualisointiin perustuvat ratkaisut ovat teh-

neet. Tällä on dramaattinen vaikutus suorituskykyyn: koska emuloinnin suorittamiseen käytetyt kellojaksot voidaan hyödyntää varsinaisen ohjelmakoodin, eli virtualisoidun palvelimen toimintojen, ajamiseen, saavutetaan laitteistoavusteisella virtualisoinnilla lähes natiivi suorituskyky. Mittausten mukaan virtualisoinnin vaikutus nykyaikaisten prosessorien suorityskykyyn onkin vain muutaman prosentin luokkaa. Sama pätee palvelimien verkkoyhteyksien virtualisoinnissa. Suoritetut mittaukset ja vertailut muihin julkaisuihin kertovat, että paravirtualisoitujen VirtIO – verkkoajureiden suorituskyky on käytännössä täysin samaa luokkaa fyysisen verkkoraudan kanssa. Virtualisoitujen levyjärjestelmien nopeudessa on vielä kuitenkin parantamisen varaa natiiviin suorituskykyyn verrattuna. Tämän ongelman kiertämiseen on käytetty kahta eri lähestymistapaa. Koska virtualisoitujen verkkoratkaisuiden suorituskyky on mittausten mukaan hyvä, voidaan palvelimissa tarvittava levytila tarjota erikseen tallennuskäyttöä varten suunnitelluilta NAS- ja SAN – laitteilta. Mikäli paikallista levytilaa kuitenkin ehdottomasti tarvitaan, toinen käytetty suorituskykyinen kiertotapa on antaa virtuaalikoneelle PCI – läpivientiä suora hallinta virtualisointialustassa olevaan RAID – korttiin ja siten samalla siihen kytkettyihin levyihin.

KVM:n käyttöönotto on nykyään myös helppoa: koska ratkaisu on nykyään osa Linux - ydintä, asennus onnistuu melko pienellä ponnistuksella. Linux on myös perinteisesti tarjonnut hyvät lähtökohdat etäkäyttöön ja todella vahvat verkko-ominaisuudet sekä on keskittynyt paitsi ytimen, myös sen päällä toimivien ohjelmiston vakauteen ja tietoturvallisuuteen.

Näin voidaankin todeta, että virtualisointi on ratkaisu esitettyihin ongelmiin, tapahtuu se sitten yritysten pienissä laitesaleissa tai pilvioperaattorin laitetiloissa. KVM:n ekosysteemi on lähtenyt valtavaan kasvuun ja tarjoaa merkittävän määrän työkaluja kaikentyyppisiin virtualisointitarpeisiin. Eikä tämä kasvu näytä ihan lähitulevaisuudessa hidastuvan, sillä yhä useampi pilvioperaattori käyttää KVM:ää virtualisointikomponenttina tarjoamissaan palveluissa.

Tälle työlle oli asetettu tavoitteeksi selvittää lukijalle KVM - palvelinvirtualisoinnissa tarvittavia tekniikoita, niiden taustoja, toimintaa sekä niihin liittyviä etuja ja mahdollisia ongelmakohtia. Mielestäni tämä onnistui riittävällä laajuudella ja tarkkuudella. Näin jälkikäteen pohdittuna leipätyöni tarjoama kokemus palvelimista ja virtualisointijärjestelmistä on tarjonnut ainakin itselleni selkeän näkökulman tässä diplomityössä esitettyihin virtualisointiteemoihin ja on myös ruokkinut riittävää kriittisyyttä mittaustuloksia, olivat ne sitten omia tai muissa julkaisussa esitettyjä, kohtaan. Toivon, että tämä kokemani elämys välittyy myös jollakin tasolla lukijalle.

LÄHTEET

(Ashley et al. 2016) W. David Ashley, Daniel Berrange, Chris Lalancette, Laine Stump, Daniel Veillard, Dani Coulson, Davin Jorm, Scott Radvan (2016), Libvirt Application Development Guide Using Python: A guide to libvirt application development with Python, version 1.1, Red Hat 2012, s. 130

(Bowker 2016) Mark Bowker (2016), Workplace Mobility, Consumerization, and Cloud Drive End-user Computing Transformation, The Enterprise Strategy Group, s. 11

(Bradkin 2009) Jon Bradkin (2009), With long history of virtualization behind it, IBM looks to the future, artikkeli NetworkWorld, http://www.networkworld.com/article/2254433/virtualization/with-long-history-of-virtualization-behind-it–ibm-looks-to-the-future.html, s. 2

(Burns 2010) Bill Burns (2010), KVM in Red Hat Enterprise Linux, Red Hat Summit 2010, s. 24

(Chen et al. 2011) Gary Chen, Al Gillen (2011), KVM for Server Virtualization: An Open Source Solution Comes of Age, IDC & IBM, s. 8

(Chierici et al. 2010) Andrea Chierici, Riccardo Veraldi (2010), A quantitative comparison between xen and kvm, 17th International Conference on Computing in High Energy and Nuclear Physics, s. 12

(Duarte 2008) Gustavo Duarte (2008), Brain food for hackers, CPU Rings, Privilege, and Protection, artikkeli, 20.8.2008, http://duartes.org/gustavo/blog/post/cpu-rings-privilege-and-protection/, s. 5

(Eidus et al. 2009) Izik Eidus, Hugh Dickins (2009), How to use the Kernel Samepage Merging feature, Linux Kernel Archives, URL: https://www.kernel.org/doc/Documentation/vm/ksm.txt, 17 Nov 2009

(Farkas et al. 2013) János Farkas, Don Fedyk, Norman Finn, Eric Gray, Michael David Johas Teener, Glenn Parsons, Panagiotis Saltsidis, Patricia Thaler (2013), IEEE 802.1Q Media Access Control Bridges and Virtual Bridged Local Area Networks, March 10, 2013, s. 77

(Gates et al. 2003) Mark Gates, Ajay Tirumala, Jon Dugan, Kevin Gibbs (2003), Iperf user Docs, s. 15

(Geer 2009) David Geer (2009), The OS Faces a Brave New World, IEEE Computer Society Press: Computer Volume 42, URL: www.eecs.umich.edu /eecs/about/articles/**2009**/IEEEXplore-2.pdf, s. 15 - 17

(Gomez-Folgar et al. 2014) Fernando Gomez-Folgar, Antonio Garcia Loureiro, Tomas Fernandez Pena, J. Isaac Zablah, Natalia Seoane (2014), Implementation of the KVM hypervisor on several cloud platfors: tuning the Apache CloudStack agent, High Performance Computing and Communications, 2014 IEEE 6th International Symposium on Cyberspace Safety and Security, s. 260 - 263

(Grattafiori 2016) Aaron Grattafiori (2016), Understanding and Hardening Linux Containers, NCC Group, s. 122

(Guangrong 2011) Xiao Guangrong (2011), KVM MMU Virtualization, s. 28

(Herrmann et al. 2016) Jiri Herrmann, Laura Bailey, Tahlia Richardson, Scott Radvan, Dayle Parker (2016), Red Hat Enterprise Linux 6 Virtualization Host Configuration and Guest Installation Guide, Red Hat Inc., s. 159

(Huffman et al. 2016) Christian Huffman, Mirek Jahoda, Jana Heves, Stephen Wadeley (2016), Red Hat Enterprise Linux 7 Networking Guide: Configuration and Administration of Networking for Red Hat Enterprise Linux 7, Red Hat, s.225

(Huynh et al. 2013) Khoa Huynh, Andrew Theurer (2013), KVM Virtualized I/O Performance: Achieving Unprecedented I/O Performance Using Virtio-Blk-Data-Plane Technology Preview in SUSE Linux Enterprise Server 11 Service Pack 3, IBM Linux Technology Center, s. 16

(IBM 2012) IBM (2012), Kernel Virtual Machine (KVM) : Best practices for KVM, Second edition, s. 48

(IEEE 802.1QTM-2014/Cor) IEEE (2015) Std 802.1QTM-2014/Cor 1-2015 Bridges and Bridged Networks, s. 122

(Jones 2010) Tim Jones (2010), Virtio: An I/O virtualization framework for Linux: Paravirtualized I/O with KVM and lguest, IBM Corporation, URL: http://www.ibm.com/developerworks/linux/library/l-libvirt/l-libvirt-pdf.pdf, s. 9

(Jones 2012) Tim Jones (2012), Managing Vms with the Virtual Machine Manager, IBM Corporation, URL: https://www.ibm.com/developerworks/cloud/library/cl-anagingvms/cl-managingvms-pdf.pdf, s. 14

(Kivity et al. 2007) Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, Anthony Liguori (2007), Kvm: the Linux Virtual Machine Monitor, Proceedings of the Linux Symposium, Volume One, June 27th-30th 2007 Ottawa (Ontario) Canada, s. 225 - 230

(Kopytov 2009) Alexey Kopytov (2009), SysBench Manual, MySQL AB, URL: http://imysql.com/wp-content/uploads/2014/10/sysbench-manual.pdf, s. 17

(Kourai et al. 2014) Kenichi Kourai, Kousuke Nakamura (2014), Efficient VM Introspection in KVM and Performance Comparison with Xen, 2014 IEEE 20th Pacific Rim International Symposium on Dependable Computing, s. 192 - 202

(Li et al. 2009) Shengzhao Li, Qinfen Hao, Limin Xiao, Qingling Xu (2009), Optimizing Network Virtualization in Kernel-based Virtual Machine, 2009 First International Conference on Information Science and Engineering, 26 – 28 December 2009, s. 282 - 285

(Libvirt 2016) Libvirt (2016), Libvirt Virtualization API, URL: https://libvirt.org/formatdomain.html, s. 29

(Liguori 2007) Anthony Liguori (2007), The Myth of Type I and Type II Hypervisors, http://blog.codemonkey.ws/2007/10/myth-of-type-i-and-type-ii-hypervisors.html, luettu 8.3.2015, artikkeli, 2007, s. 2

(Linux Foundation 2009) Linux Foundation (2009), Bridge, URL: http://www.linuxfoundation.org/collaborate/workgroups/networking/bridge, noudettu 22.3.2016, s. 9

(Marr et al. 2002) Deborah T. Marr, Frank Binns, David L. Hill, Glenn Hinton, David A. Koufaty, J. Alan Miller, Michael Upton (2002), Hyper-Threading Technology Architecture and Microacrhitecture, Intel Technology Journal Q1, 2002, s. 12

(Meier 2008) Shannon Meier (2008), IBM Systems Virtualization: Servers, Storage, and Software, IBM Redbook REDP-4396-00, s. 96

(Milberg et al. 2013) Ken Milberg, Manish Bhardwaj, Barry Cohen (2013), Server Virtualization on UNIX Systems: A comparison between HP Integrity Servers with HP-UX and IBM Power Systems with AIX, Edison Group, s. 26

(Noyes 2013) Katherine Noyes (2013), Docker: A Shipping container for Linux Code, Linux.com verkkojulkaisusta 1.8.2013, s. 3

(Parker et al. 2015) Dayle Parker, Scott Radvan (2015), Red Hat Enterprise Linux 7 Virtualization Tuning and Optimization Guide, s. 50

(Pfaff et al. 2015) Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan J. Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Jonathan Stringer, Pravin Shelar, Keith Amidon, Martín Casado (2015), The Design and Implementation of Open vSwitch, Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI ’15), May 4-6 2015, Oakland, CA, USA, s. 117 - 130

(Qemu 2016) QEMU Emulator User Documentation (2016), http://wiki.qemu.org/download/qemu-doc.html, s. 50

(Red Hat 2015) Red Hat (2015), Kernel based virtual machine, s. 11

(Reshetova et al. 2014) Elena Reshetova, Janne Karhunen, Thomas Nyman, N. Asokan (2014), Security of OS-level virtualization technologies, Secure IT Systems: 19th Nordic Conference, NordSec 2014, Troms, Norway, October 15-17, 2014, Proceedings", s. 77 - 93

(Rizzo et al. 2013) Luigi Rizzo, Giuseppe Lettieri, Vincenzo Maffione (2013), Speeding Up Packet I/O in Virtual Machines, Proceedings of the Ninth ACM/IEEE Symposium on Architectures for Networking and Communications Systems 2013, San Jose, California, USA, s. 47-58

(Ruest et al. 2009) Danielle Ruest, Nelson Ruest (2009), Virtualization: A Beginner's Guide, McGraw-Hill Companies Network Professional's library, ISBN: 978-0-07-161402-3, s. 463

(Russell 2008) Rusty Russell (2008), Virtio: Towards a De-Facto Standard For Virtual I/O Devices, SIGOPS Operating Systems Review – Research and developments in the Linux Kernel Volume 42 Issue 5 July 2008, s. 95 - 103

(Soriga et al. 2013) Stefan Soriga, Mihai Barbulescu (2013), A comparison of the performance and scalability of Xen and KVM hypervisors, Networking in Education and Research, 2013 RoEduNet International Conference 12th Edition, 26-28 September 2013, s. 1 - 6

(Stallings 2012) William Stallings (2012), Operating Systems – Internals and design principles, 7th edition, 2012, Prentice Hall, ISBN-13: 978-0-13-230998-1, s. 820

(VMWare 2007) VMWare (2007), Understanding Full Virtualization, Paravirtualization, and Hardware Assist, white paper, s. 17

(VMWare 2009) VMWare (2009), Software and Hardware Techniques for x86 Virtualization, s. 9

(Walters et al. 2014) John Paul Walters, Andrew J. Younge, Dong-In Kang, Ke-Thia Yao, Mikyung Kang, Stephen P. Crago, Geoffrey C. Fox (2014), GPU Passthrough Performance: A Comparison of KVM, Xen, VMWare ESXi, and LXC for CUDA and OpenCL Applications, 2014 IEEE 7th International Conference on Cloud Computing, s. 636 - 643

(Wang et al. 2010) Jiang Wang, Sameer Niphadkar, Angelos Stavrou, Anup K. Ghosh (2010), A Virtualization Architecture for In-depth Kernel Isolation, 2014 47th Hawaii International Conference on System Sciences, ISBN: 978-0-7695-3869-3, s. 1 - 10

(Xen 2016) Xen (2016), Xen, history, Linux foundation Collaborative Projects: Xenproject.org, URL: http://www.xenproject.org/about/history.html, luettu 9.2.2016, s. 3

(Zhang et al. 2010) Binbin Zhang, Xiaolin Wang, Rongfeng Lai, Liang Yang, Zhenlin Wang, Yingwei Luo (2010), Evaluating and Optimizing I/O Virtualization in Kernel-based Virtual Machine (KVM), Network and Parallel Computing: IFIP International Conference, NPC 2010, Zhengzhou, China, September 13-15, 2010, s. 220 - 231

Liite A: KVM-Qemu, prosessorituki ja tunnistetut CPU-ominaisuudet

x86	qemu64	QEMU Virtual CPU version 2.0.0
x86	phenom	AMD Phenom(tm) 9550 Quad-Core Processor
x86	core2duo	Intel(R) Core(TM)2 Duo CPU T7700 @ 2.40GHz
x86	kvm64	Common KVM processor
x86	qemu32	QEMU Virtual CPU version 2.0.0
x86	kvm32	Common 32-bit KVM processor
x86	coreduo	Genuine Intel(R) CPU T2600 @ 2.16GHz
x86	486	
x86	pentium	
x86	pentium2	
x86	pentium3	
x86	athlon	QEMU Virtual CPU version 2.0.0
x86	n270	Intel(R) Atom(TM) CPU N270 @ 1.60GHz
x86	Conroe	Intel Celeron_4x0 (Conroe/Merom Class Core 2)
x86	Penryn	Intel Core 2 Duo P9xxx (Penryn Class Core 2)
x86	Nehalem	Intel Core i7 9xx (Nehalem Class Core i7)
x86	Westmere	Westmere E56xx/L56xx/X56xx (Nehalem-C)
x86	SandyBridge	Intel Xeon E312xx (Sandy Bridge)
x86	Haswell	Intel Core Processor (Haswell)
x86	Opteron_G1	AMD Opteron 240 (Gen 1 Class Opteron)
x86	Opteron_G2	AMD Opteron 22xx (Gen 2 Class Opteron)
x86	Opteron_G3	AMD Opteron 23xx (Gen 3 Class Opteron)
x86	Opteron_G4	AMD Opteron 62xx class CPU
x86	Opteron_G5	AMD Opteron 63xx class CPU
x86	host	KVM processor with all supported host features (only available in KVM mode)

Recognized CPUID flags:
pbe ia64 tm ht ss sse2 sse fxsr mmx acpi ds clflush pn pse36 pat cmov mca pge mtrr sep apic cx8 mce pae msr tsc pse de vme fpu hypervisor rdrand f16c avx osxsave xsave aes tsc-deadline popcnt movbe x2apic sse4.2|sse4_2 sse4.1|sse4_1 dca pcid pdcm xtpr cx16 fma cid ssse3 tm2 est smx vmx ds_cpl monitor dtes64 pclmulqdq|pclmuldq pni|sse3 smap adx rdseed rtm invpcid erms bmi2 smep avx2 hle bmi1 fsgsbase 3dnow 3dnowext lm|i64 rdtscp pdpe1gb fxsr_opt|ffxsr mmxext nx|xd syscall perfctr_nb perfctr_core topoext tbm nodeid_msr tce fma4 lwp wdt skinit xop ibs osvw 3dnowprefetch misalignsse sse4a abm cr8legacy extapic svm cmp_legacy lahf_lm pmm-en pmm phe-en phe ace2-en ace2 xcrypt-en xcrypt xstore-en xstore kvm_pv_unhalt kvm_pv_eoi kvm_steal_time kvm_asyncpf kvmclock kvm_mmu kvm_nopiodelay kvmclock pfthreshold pause_filter decodeassists flushbyasid vmcb_clean tsc_scale nrip_save svm_lock lbrv npt

Liite B:

KVM API, yksinkertaisen virtuaalikoneen luominen 16-bittisen x86-koodin ajamista varten.

```
1       /* Sample code for /dev/kvm API
2        *
3        * Copyright (c) 2015 Intel Corporation
4        * Author: Josh Triplett <josh@joshtriplett.org>
5        *
6        * Permission is hereby granted, free of charge, to any person
        obtaining a copy
7        * of this software and associated documentation files (the
        "Software"), to
8        * deal in the Software without restriction, including without
        limitation the
9        * rights to use, copy, modify, merge, publish, distribute,
        sublicense, and/or
10       * sell copies of the Software, and to permit persons to whom the
        Software is
11       * furnished to do so, subject to the following conditions:
12       *
13       * The above copyright notice and this permission notice shall be
        included in
14       * all copies or substantial portions of the Software.
15       *
16       * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY
        KIND, EXPRESS OR
17       * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
        MERCHANTABILITY,
18       * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
        EVENT SHALL THE
19       * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES
        OR OTHER
20       * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
        OTHERWISE, ARISING
21       * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
        OTHER DEALINGS
22       * IN THE SOFTWARE.
23       */
24      #include <err.h>
25      #include <fcntl.h>
26      #include <linux/kvm.h>
27      #include <stdint.h>
28      #include <stdio.h>
29      #include <stdlib.h>
30      #include <string.h>
31      #include <sys/ioctl.h>
32      #include <sys/mman.h>
33      #include <sys/stat.h>
34      #include <sys/types.h>
35
36      int main(void)
37      {
38          int kvm, vmfd, vcpufd, ret;
39          const uint8_t code[] = {
40              0xba, 0xf8, 0x03, /* mov $0x3f8, %dx */
41              0x00, 0xd8,       /* add %bl, %al */
42              0x04, '0',        /* add $'0', %al */
43              0xee,             /* out %al, (%dx) */
44              0xb0, '\n',       /* mov $'\n', %al */
45              0xee,             /* out %al, (%dx) */
46              0xf4,             /* hlt */
47          };
48          uint8_t *mem;
```

```
49          struct kvm_sregs sregs;
50          size_t mmap_size;
51          struct kvm_run *run;
52
53          kvm = open("/dev/kvm", O_RDWR | O_CLOEXEC);
54          if (kvm == -1)
55              err(1, "/dev/kvm");
56
57          /* Make sure we have the stable version of the API */
58          ret = ioctl(kvm, KVM_GET_API_VERSION, NULL);
59          if (ret == -1)
60              err(1, "KVM_GET_API_VERSION");
61          if (ret != 12)
62              errx(1, "KVM_GET_API_VERSION %d, expected 12", ret);
63
64          vmfd = ioctl(kvm, KVM_CREATE_VM, (unsigned long)0);
65          if (vmfd == -1)
66              err(1, "KVM_CREATE_VM");
67
68          /* Allocate one aligned page of guest memory to hold the
        code. */
69          mem = mmap(NULL, 0x1000, PROT_READ | PROT_WRITE, MAP_SHARED |
        MAP_ANONYMOUS, -1, 0);
70          if (!mem)
71              err(1, "allocating guest memory");
72          memcpy(mem, code, sizeof(code));
73
74          /* Map it to the second page frame (to avoid the real-mode
        IDT at 0). */
75          struct kvm_userspace_memory_region region = {
76              .slot = 0,
77              .guest_phys_addr = 0x1000,
78              .memory_size = 0x1000,
79              .userspace_addr = (uint64_t)mem,
80          };
81          ret = ioctl(vmfd, KVM_SET_USER_MEMORY_REGION, &region);
82          if (ret == -1)
83              err(1, "KVM_SET_USER_MEMORY_REGION");
84
85          vcpufd = ioctl(vmfd, KVM_CREATE_VCPU, (unsigned long)0);
86          if (vcpufd == -1)
87              err(1, "KVM_CREATE_VCPU");
88
89          /* Map the shared kvm_run structure and following data. */
90          ret = ioctl(kvm, KVM_GET_VCPU_MMAP_SIZE, NULL);
91          if (ret == -1)
92              err(1, "KVM_GET_VCPU_MMAP_SIZE");
93          mmap_size = ret;
94          if (mmap_size < sizeof(*run))
95              errx(1, "KVM_GET_VCPU_MMAP_SIZE unexpectedly small");
96          run = mmap(NULL, mmap_size, PROT_READ | PROT_WRITE,
        MAP_SHARED, vcpufd, 0);
97          if (!run)
98              err(1, "mmap vcpu");
99
100         /* Initialize CS to point at 0, via a read-modify-write of
        sregs. */
101         ret = ioctl(vcpufd, KVM_GET_SREGS, &sregs);
102         if (ret == -1)
103             err(1, "KVM_GET_SREGS");
104         sregs.cs.base = 0;
105         sregs.cs.selector = 0;
106         ret = ioctl(vcpufd, KVM_SET_SREGS, &sregs);
107         if (ret == -1)
108             err(1, "KVM_SET_SREGS");
```

```
109
110         /* Initialize registers: instruction pointer for our code,
    addends, and
111          * initial flags required by x86 architecture. */
112         struct kvm_regs regs = {
113             .rip = 0x1000,
114             .rax = 2,
115             .rbx = 2,
116             .rflags = 0x2,
117         };
118         ret = ioctl(vcpufd, KVM_SET_REGS, &regs);
119         if (ret == -1)
120             err(1, "KVM_SET_REGS");
121
122         /* Repeatedly run code and handle VM exits. */
123         while (1) {
124             ret = ioctl(vcpufd, KVM_RUN, NULL);
125             if (ret == -1)
126                 err(1, "KVM_RUN");
127             switch (run->exit_reason) {
128             case KVM_EXIT_HLT:
129                 puts("KVM_EXIT_HLT");
130                 return 0;
131             case KVM_EXIT_IO:
132                 if (run->io.direction == KVM_EXIT_IO_OUT && run-
    >io.size == 1 && run->io.port == 0x3f8 && run->io.count == 1)
133                     putchar(*(((char *)run) + run->io.data_offset));
134                 else
135                     errx(1, "unhandled KVM_EXIT_IO");
136                 break;
137             case KVM_EXIT_FAIL_ENTRY:
138                 errx(1, "KVM_EXIT_FAIL_ENTRY:
    hardware_entry_failure_reason = 0x%llx",
139                      (unsigned long long)run-
    >fail_entry.hardware_entry_failure_reason);
140             case KVM_EXIT_INTERNAL_ERROR:
141                 errx(1, "KVM_EXIT_INTERNAL_ERROR: suberror = 0x%x",
    run->internal.suberror);
142             default:
143                 errx(1, "exit_reason = 0x%x", run->exit_reason);
144             }
145         }
146     }
```