Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Marko Đukić

Primerjava zmogljivosti virtualizacijskih tehnologij in tehnologij vsebnikov

magistrsko delo

študijski program
računalništvo in informatika

Mentorica: doc. dr. Mojca Ciglarič

Ljubljana, 2016

Zahvaljujem se mentorici doc. dr. Mojci Ciglarič za pomoč, vodenje in potrpežljivost pri izdelavi magistrskega dela in as. dr. Matjažu Pančurju za nasvete glede testiranja, virtualizacije in tehnologije vsebnikov.
Hvala tudi skupnosti uporabnikov Xen za koristno pomoč pri uporabi in nastavitvah programske rešitve Xen.

Delo posvečam svoji ženi Špeli.

Hvala za spodbudo in za potrpljenje,
brez tebe mi ne bi uspelo!

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
BT	binary translator	binarni prevajalnik
CPE	central processing unit	centralna procesna enota
IaaS	Infrastructure-as-a-Service	infrastruktura kot storitev
KVM	Kernel Virtual Machine	navidezni stroj jedra
MMU	memory managment unit	enota za upravlj. pomnilnika
NUMA	non-uniform memory architecture	neenotna arhitektura pomnilnika
OS	operating system	operacijski sistem
PC	personal computer	osebni računalnik
PaaS	Platform-as-a-Service	platforma kot storitev
SaaS	Software-as-a-Service	programska oprema kot storitev
VE	virtual environment	navidezno okolje
VPS	virtual private server	navidezni zasebni strežnik
VM	virtual machine	navidezni stroj
VMCB	virtual machine control block	nadzorni blok navid. stroja
VMCS	virtual machine control structure	nadzorna struktura navid. stroja
VMM	virtual machine monitor	nadzornik navideznih strojev
TLB	translation lookaside buffer	medpomnilnik MMU

Povzetek

Naslov: Primerjava zmogljivosti virtualizacijskih tehnologij in tehnologij vsebnikov

Magistrsko delo obravnava primerjavo virtualizacije in tehnologije vsebnikov s ciljem določitve področij, na katerih ima uporaba določene tehnologije prednost pred drugo. Arhitektura x86 v svojih zametkih ni bila mišljena za uporabo v virtualizaciji – zaradi "na virtualizacijo občutljivih" ukazov pri tej arhitekturi ni možna rešitev vrste "ujemi in emuliraj", ki je uporabljena v drugih arhitekturah. Razvijalci rešitev za virtualizacijo so zato morali razviti drugačne rešitve, kot sta polna virtualizacija z uporabo binarne translacije in paravirtualizacija, po letu 2005 pa so proizvajalci procesorjev virtualizaciji dodali tudi strojno podporo. Vse tri omenjene rešitve za poganjanje navideznih strojev zahtevajo določene stroške režije. Tehnologija vsebnikov, ki jo imenujemo tudi virtualizacija na nivoju operacijskega sistema ali lahkokategorna virtualizacija, se tem stroškom izogne, saj vsebnik za izolacijo aplikacij uporablja funkcionalnosti gostiteljskega operacijskega sistema. Večji pomanjkljivosti tehnologije vsebnikov sta večje varnostno tveganje, ker si vsebniki delijo isto jedro operacijskega sistema, in nezmožnost poganjanja drugih operacijskih sistemov znotraj vsebnika.
Vsebniki naj bi tako gostitelja obremenjevali manj kot navidezni stroji, posledično to pomeni, da je lahko za vsebnike na voljo več virov, kar pomeni boljše performančne zmogljivosti. Zanimalo nas je, kolikšna je ta razlika v zmogljivosti med vsebnikom in navideznim strojem, zato smo izvedli več testiranj na različnih področjih, pri čemer smo teste izvedli najprej na fizičnem sistemu, nato pa še na izbranih rešitvah: KVM, Xen, LXC in Docker. Izvajali smo naslednje teste: Intel Optimized LINPACK, STREAM, RandomAccess, netperf, fio, bonnie++ in UnixBench.

Rezultati so pokazali rahlo prednost tehnologije vsebnikov pred virtualizacijo, a ne na vseh področjih. Prednost tudi ni dovolj velika, da bi lahko brez zadržkov določili področja, na katerih ima uporaba neke tehnologije absolutno prednost pred drugo. Večje razlike v rezultatih smo zasledili tudi pri različnih rešitvah iste tehnologije, kar kaže na to, da implementacija rešitve vpliva na zmogljivost.

Ključne besede: virtualizacija, tehnologija vsebnikov, hipervizor, navidezni stroj, vsebnik, storitve v oblaku.

Abstract

Title: Performance comparison of virtualization and container technologies

In this work we compare virtualization and container technologies with the goal of recognizing fields where one technology has advantages over the other. In its beginnings the x86 architecture was not designed with virtualization in mind. The existence of "virtualization sensitive" instructions does not allow the use of "trap-and-emulate" solutions, as they are used in other architectures. As a consequence, other solutions were developed, such as full virtualization with binary translation and paravirtualization, with hardware support for virtualization added to processors after 2005. All three solutions add overhead when running virtual machines. Container technology, also called operating-system-level virtualization or lightweight virtualization, avoids this overhead because containers use operating system features for application isolation. Main shortcomings of container technology are a bigger security risk because of the shared kernel and an inability to run other operating systems inside a container.

Containers should present a smaller load for the host, consequently that means more resources available for containers, which means better performance. We wanted to research how large this difference in performance between a container and a virtual machine is, so we executed tests covering different fields. Firstly, we tested the physical system and then we tested different virtualization and containerization solutions: KVM, Xen, LXC and Docker. Our chosen tests were: Intel Optimized LINPACK, STREAM, RandomAccess, netperf, fio, bonnie++ and UnixBench.

The results show a slight advantage in performance for containers in all fields. This advantage, however, is not large enough for us to conclude without reservation in which field a certain technology has an absolute advantage before the other.

There were also differences in results using different solutions of the same technology, which show the effect of implementation of the solution on the performance.

Keywords: virtualization, container technology, hypervisor, virtual machine, container, cloud computing.

Poglavje 1

Uvod

V delu smo primerjali virtualizacijo in tehnologijo vsebnikov, dve tehnologiji, ki omogočata, da en fizični računalnik razdelimo na več med seboj izoliranih logičnih enot – navideznih strojev oziroma vsebnikov. Poleg fleksibilnosti, ki jo prinaša sočasno poganjanje več (tudi različnih) operacijskih sistemov, lahko uporaba teh tehnologij poveča učinkovitost izrabe virov gostiteljskega računalnika.
Virtualizacija je nepogrešljivo orodje pri storitvah v oblaku, saj njena uporaba omogoča marsikatero funkcionalnost te vrste storitev. V zadnjih nekaj letih pa se kot posledica razvoja določenih funkcionalnosti Linuxovega jedra kot alternativa virtualizaciji (tudi pri storitvah v oblaku) pojavlja tehnologija vsebnikov. Prednost tehnologije vsebnikov, ki jo imenujemo tudi virtualizacija na nivoju operacijskega sistema ali tudi lahkokategorna virtualizacija, naj bi bila v manjših stroških režije v primerjavi z virtualizacijo in s tem posledično manjši obremenitvi gostiteljskega sistema.
Cilj tega dela je bil ugotoviti, kolikšna je razlika v performančnih zmogljivostih med navideznim strojem in vsebnikom na različnih področjih ter, ali se na podlagi te razlike lahko nedvoumno odločimo za uporabo določene tehnologije pred drugo.

1.1 Sorodne raziskave

Področja, kot so storitve v oblakih, virtualizacija in tehnologija vsebnikov, so obravnavana v precej literature, navajali smo jo v poglavjih, ki pokrivajo omenjena področja.
V delu [10] avtorji obravnavajo primerjavo med rešitvama KVM in Docker. S testi so pokrili različna področja (računsko, podatkovno, mrežno in tudi dve aplikativni), rešitvi so testirali na večprocesorskem strežniku z arhitekturo NUMA. Podobno v delu [19] obravnavajo primerjavo med rešitvami KVM, Docker in LXC. Njihovi testi pokrivajo tudi podobna področja (računsko, podatkovno in mrežno), manjka pa aplikativni test ali test vpliva sistemskih klicev na performančne zmogljivosti. Ugotovitve obeh del kažejo prednost vsebnikov pred navideznimi stroji, obenem pa zaznavajo občutno izboljšanje rešitve KVM v zadnjih letih. V obeh delih pogrešamo vključitev tudi drugih rešitev za virtualizacijo v primerjavo za širši pregled stanja virtualizacije.
V delu [13] so naredili malce drugačno primerjavo, in sicer so primerjali navidezni stroj znotraj Amazonovega "oblaka" (*AWS ec2)* in vsebnik v Dockerju, nameščenem na dva fizična sistema. Primerjali so odziva obeh sistemov na vedno več zahtev in skalabilnost sistemov s povečevanjem obremenitve CPE (pri tem so izkoristili samodejno skaliranje, ki ga omogoča AWS ec2). Rezultati kažejo na boljše performančne zmogljivosti Dockerjevih vsebnikov, a je res, da moramo upoštevati, da so navidezni stroji tekli v oblaku in ne na namenskih računalnikih tako kot vsebniki, kar lahko vpliva na zmogljivost.

1.2 Zgradba nadaljnjega besedila

Poglavje 2 zajema pregled storitev v oblaku, prikazani so formalne definicije in modeli, njihova povezava z virtualizacijo in tehnologijo vsebnikov ter to, kakšna so tveganja pri uporabi. Poglavje 3 predstavlja virtualizacijo, zgodovino razvoja, problematiko virtualizacije arhitekture x86 in načine reševanja te problematike: polno virtualizacijo z uporabo binarne translacije, strojno podprto virtualizacijo in paravirtualizacijo. Na koncu sledi še predstavitev nekaterih popularnejših rešitev za virtualizacijo. V poglavju 4 prikazujemo tehnologijo vsebnikov v operacijskem sistemu Linux in podrobneje, kako je izvedena v rešitvah LXC in Docker. V po-

glavju 5 sledi primerjava obeh tehnologij, njunih funkcionalnosti, zmogljivosti in tveganj. Prikaz testiranja obeh tehnologij sledi v poglavju 6. Najprej opišemo način testiranja, izbrane rešitve in izbrana orodja za testiranje, predstavitev rezultatov pa dopolnjuje še diskusija. Magistrsko delo sklenemo v poglavju 7.

Poglavje 2

Storitve v oblaku

Že dalj časa se računalniške storitve z osebnih računalnikov in lokalnih strežniških sistemov selijo na splet. Klasičnim spletnim storitvam, kot so elektronska pošta, spletne strani in hramba podatkov, so se pridružile mnoge nove, med drugim tudi storitve, ki so bile prvotno namenjene osebnim računalnikom (npr. urejanje besedil in fotografij), pri čemer vsak dan nastajajo in ugašajo storitve, namenjene temu, kar zahteva trg. Podjetja prehajajo iz modela enkratne prodaje programske opreme, ki jo uporabnik namesti na osebni računalnik, na naročniški model. Uporabnik tam plačuje mesečno ali letno naročnino (lahko tudi po porabi), za kar dobi na voljo za uporabo izbrano storitev, pri čemer mu ni treba skrbeti za namestitev, vzdrževanje in ažurnost.

Take storitve so del storitev, ki jih poimenujemo "storitve v oblaku" (bolj formalne definicije sledijo v nadaljevanju). Omenjamo jih, ker je vzrok za njihovo razširjenost tudi razvoj virtualizacije. Mnoge lastnosti storitev v oblaku ne bi bile možne ali bi jih težje realizirali brez uporabe virtualizacije. Uporaba virtualizacije omogoča dinamično prilagajanje zahtevam uporabnikov in obremenitvam sistema, strojno opremo lahko izkoristimo učinkoviteje, stroški postavitve in vzdrževanja sistema storitev v oblaku so nižji itn.

Poleg virtualizacije moramo omeniti, da je v porastu (tudi) pri storitvah v oblaku uporaba tehnologije vsebnikov namesto virtualizacije ali komplementarno z njo. Tehnologija vsebnikov omogoča podobno funkcionalnost kot virtualizacija, le realizirano na drugačen način.

2.1 Kaj so storitve v oblaku

Angleški izraz *cloud computing* lahko dobesedno prevedemo kot računanje v oblaku, v slovenščini pa bi izrazu bolje ustrezala besedna zveza *storitve v oblaku*. Tu gre namreč za množico različnih storitev, od uporabe navideznih strojev, hrambe podatkov do že pripravljenih aplikacij, ki "tečejo v oblaku".
Pojem je postal priljubljen po objavi skupne iniciative IBM-a in Googla leta 2007 [12, 16]. Podjetji sta takrat predstavili idejo o postavitvi velikih podatkovnih centrov z ogromno računsko močjo, ki bi bili na voljo raziskovalnim ustanovam in študentom, do njih pa bi dostopali preko interneta.
Kaj sploh je oblak oz. storitve v oblaku? Na spletu najdemo ogromno definicij. Če bi gledali zelo široko, bi lahko oblak definirali kot storitev ali množico storitev na oddaljenem računalniku, do katere/-ih dostopamo preko interneta. Ker je taka definicija preveč splošna, saj dejansko opisuje kar koli, kar najdemo na spletu, v nadaljevanju podajamo bolj formalizirane definicije.
Storitve v oblaku so naslednji korak v razvoju distribuiranega računalništva in paralelnega procesiranja. Oblak lahko opišemo kot zbirko različnih računskih, podatkovnih, mrežnih in drugih virov, ki so na voljo uporabnikom v obliki storitev preko protokola TCP/IP. Vzpostavljen je s standardnimi protokoli za povezovanje (*interconnect*) in preizkušenimi tehnologijami, ki se uporabljajo v podatkovnih centrih.

2.2 Modeli storitev v oblaku

Ameriški Nacionalni inštitut za standarde in tehnologijo (*National Institute for Standards and Technology – NIST*) je podal dokument [17], v katerem so definirali sestavo modela storitev v oblaku: karakteristike, storitveni model in postavitveni model. Mnoge karakteristike storitev v oblaku so možne prav zaradi uporabe virtualizacije.

2.2.1 Karakteristike storitev v oblaku

Storitve v oblaku imajo vsaj naslednje karakteristike:

- Avtomatizirana storitev na zahtevo (*On-demand self-service*): Stranka lahko

rezervira oblačne vire po potrebi in avtomatizirano, brez posredovanja ponudnika storitev.

- Širokopasovni mrežni dostop (*Broad network access*): Vsi viri so dostopni preko omrežja z uporabo standardiziranih mehanizmov.
- Združevanje virov (*Resource pooling*): Viri, ki jih ponuja ponudnik storitev v oblaku (računski, omrežni in podatkovni), so združeni in hkrati na voljo več strankam, dinamično in potrebi. Stranke načeloma ne vedo, kje so viri locirani, lahko pa podajo zahteve glede lokacije, npr. država, območje, podatkovni center.
- Hitrost in elastičnost (*Rapid elasticity*): Vire ponudnik dodaja in odvzema elastično (prilagodljivo glede na zahteve), lahko tudi samodejno. Prilagoditev velikostim zahtevam (skaliranje – *scaling*) je hitra, strankam se zdi, da imajo za rezervacijo na voljo neomejeno veliko virov.
- Merjena storitev (*Measured service*): Sistemi za storitve v oblaku samodejno nadzorujejo in optimizirajo vire z merjenjem porabe glede na vrsto storitve (procesiranje, hramba podatkov, pasovna širina in aktivni uporabniki). Da je delovanje transparentno tako za uporabnika kot ponudnika storitev, nadzorni sistemi podatke o porabi virov hranijo, nadzorujejo ter poročajo upraviteljem in uporabnikom.

2.2.2 Storitveni model

Glede na nivo abstrakcije arhitekture oziroma glede na razmerje programska oprema — strojna oprema lahko storitve v oblaku na široko delimo v tri kategorije [11, 18, 27]. Pri čemer je treba vedeti, da meje med posameznimi kategorijami niso strogo določene, še več, različni ponudniki storitev v oblaku si lahko spodaj opisane pojme razlagajo po svoje. Kljub temu spodnji pojmi prispevajo k ločevanju različnih rešitev storitev v oblaku, četudi njihovi ponudniki ne uporabljajo popolnoma enakih definicij.

- Programska oprema kot storitev (*Software-as-a-Service, SaaS*): Uporabnikom so kot storitev na voljo aplikacije, do katerih dostopajo preko interneta bodisi s spletnim brskalnikom bodisi preko programskega vmesnika.

Uporabniki tu nimajo pregleda in nadzora nad oblačno infrastrukturo ter lahko imajo omejene možnosti nastavitve aplikacij, ki jih uporabljajo. Cilj SaaS je nadomestiti aplikacije na osebnih računalnikih s spletnimi, ki jih ni treba nameščati, nadgradnja pa je samodejna. Namesto nakupa programske opreme, ki je ponavadi dražji, imamo pri tem modelu naročnino ali plačilo po porabi, kar lahko pomeni manjše stroške. Glavna slabost takega najema aplikacij so morebitne težave, povezane z mrežno povezavo (omejen ali onemogočen dostop zaradi obremenitev ali izpada omrežja).

- Platforma kot storitev (*Platform-as-a-Service, PaaS*): Uporabnikom so na voljo programski jeziki, knjižnice in razvojna orodja – torej celotno razvojno okolje, v katerem lahko razvijejo lastne aplikacije, ki jih potem tudi namestijo v isti oblak za končne uporabnike. Uporabniki tako kot pri SaaS, nimajo nadzora nad oblačno infrastrukturo, nadzorujejo in konfigurirajo pa lahko nameščene aplikacije. Prednosti PaaS so zmanjšanje stroškov, povezanih z razvojem in namestitvijo aplikacij, možnost plačila po porabi, hitra razširljivost (*scalability*), varnost in zanesljivost.

- Infrastruktura kot storitev (*Infrastructure-as-a-Service, IaaS*): Uporabnikom so kot storitev z uporabo virtualizacije na voljo različni računski, hrambeni in omrežni viri, ki jih uporabljajo za lastne potrebe (namestitev operacijskih sistemov in aplikacij). Najpogosteje gre tu za najem oz. rezervacijo navideznega stroja, umeščenega v navidezno omrežje, na katerega se namesti izbran operacijski sistem. Kot pri SaaS in PaaS uporabniki nimajo nadzora nad oblačno infrastrukturo, vendar imajo nadzor nad operacijskimi sistemi, hrambo podatkov, aplikacijami in omejen nadzor nad omrežnimi viri. In če je pri SaaS in PaaS uporaba virtualizacije pri izvedbi skrita končnemu uporabniku, je pri IaaS dejansko del ponudbe (seveda z omejitvami). Uporaba IaaS lahko zmanjša stroške nakupa strežnikov in pripadajoče omrežne infrastrukture, kar lahko zelo pomaga predvsem novonastalim podjetjem, ki še nimajo ustreznega kapitala. V takem primeru uporabniki najamejo virtualno infrastrukturo v oblaku, ki jo prilagodijo po svojih željah in jo prilagajajo tudi med uporabo. Tudi druge prednosti so podobne kot pri PaaS.

Ponudniki storitev v oblaku ponavadi niso omejeni le na en storitveni model. Microsoftov Azure [48] na primer po zgornji definiciji ponuja tako PaaS kot IaaS, Amazon pa daje uporabnikom s svojim sistemom Amazon Web Services [29] na voljo vse tri vrste storitev.
Glede na napisano bi lahko trdili, da je SaaS nadpomenka za preostali kategoriji, saj je v vsakem primeru storitev neka programska oprema (razlika je kot rečeno le v abstrakciji). A je res, da morda to bolj drži gledano s stališča ponudnika storitev, ki pozna podrobnosti implementacije svojega "oblaka". Z uporabniškega stališča, ki ga podrobnosti delovanja ne zanimajo oziroma mu niso vidne, pa so kategorije le strožje ločene med seboj.

2.2.3 Namestitveni model

Glede na tip namestitve ločimo več modelov oblačne infrastrukture [8, 27]:

- Zasebni oblak (*private cloud*): Zasebni oblak je namenjen uporabi v zasebnem podjetju ali organizaciji. Lahko je nameščen lokalno, lahko tudi pri nekem zunanjem ponudniku storitev. Upravljanje oblaka je pod nadzorom podjetja ali organizacije, kar lahko v primerjavi z javnim oblakom prinese izboljšave glede varnosti, zanesljivosti, kakovosti storitev ipd. Po drugi strani lahko zasebni oblak pomeni tudi višje stroške vzpostavitve, upravljanja in vzdrževanja sistema.

- Skupnostni oblak (*community cloud*): Oblak te vrste je pod nadzorom več različnih organizacij, ki si delijo skupne interese na enem ali več področjih (npr. varnost, zasebnost ipd.). Člani teh organizacij si delijo podatke in aplikacije v oblaku. Cilj take namestitve je znižanje stroškov v primerjavi z namestitvijo zasebnega oblaka in zmanjšanje tveganj, povezanih z javnimi oblaki.

- Javni oblak (*public cloud*): V javnem oblaku pod upravljanjem enega ponudnika storitev si vire deli več uporabnikov, ki za njih plačujejo po porabi (seveda obstajajo tudi neplačljive rešitve, ki financiranje pridobivajo npr. preko oglasov). Storitve javnega oblaka so ponavadi lahko dostopne, stroškovno ugodne in razširljive. Deljena uporaba virov pa lahko prinaša

tudi skrite pasti, kot so varnostna tveganja, zakonodajne neskladnosti in nezagotovljena kakovost storitev (*Quality of Service*, *QoS*).

- Hibridni oblak (*hybrid cloud*): Tu gre za kombinacijo dveh ali več vrst postavitev oblakov, ki so sicer vzpostavljeni ločeno in povezani s tehnologijami, ki omogočajo prenose podatkov in aplikacij med njimi. Primer uporabe bi bil v podjetju, ki zaradi manjših stroškov obratovanja za poslovno kritične naloge uporablja zasebni oblak, za manj kritične naloge pa javni oblak.

2.3 Zasebnost in varnost pri uporabi storitev v oblaku

Živimo v času, ko si življenja brez dostopa do svetovnega spleta skoraj ne moremo predstavljati. V zadnjem času je razširjenost uporabe "pametnih" mobilnih telefonov povzročila, da so uporabniki na splet povezani pravzaprav 24 ur na dan. Pri tej vseprisotni povezljivosti zato ne smemo pozabiti na svojo zasebnost in varnost svojih podatkov. Kljub mnogim opozorilom različnih organizacij uporabniki vse prevečkrat pozabijo na varno brskanje po spletu. Pri tem ni pomembna samo zaščita pred zlonamerno programsko opremo (*malware*), ampak tudi zaščita zasebnosti. Osebni podatki so postali tržno blago, ki ga oglaševalci kupujejo od podjetij, ki jih zbirajo na različne načine – največkrat svoja življenja razgrnejo uporabniki sami na različnih socialnih omrežjih.
Del te vseprisotne povezljivosti so tudi storitve v oblaku, sploh glede mobilnih naprav, pri katerih se ogromno podatkov shranjuje "v oblak". Podatki se tako hranijo in obdelujejo na eni ali več oddaljenih lokacijah, na različnih strežnikih. To prinese s seboj tveganja glede zasebnosti in varnosti [20], ki jih morajo upoštevati ne samo končni uporabniki storitev v oblaku, temveč tudi podjetja, ki oblak uporabljajo kot platformo za svoje storitve.

2.3.1 Tveganja, povezana z zasebnostjo

Za končnega uporabnika zasebnost pomeni, da so njegovi osebni podatki ustrezno zaščiteni pred nepooblaščenimi dostopi in zlorabami, uporaba osebnih podatkov

v poslovne namene pa mora biti v skladu z uporabnikovimi pričakovanji. Podjetja morajo z osebnimi podatki ravnati glede na zakonodajo in druga pravila, ki določajo ravnanje z njimi. Različna tveganja se pojavijo, ko storitve v oblaku zbirajo, hranijo in obdelujejo osebne podatke ter jih morda tudi posredujejo tretjim osebam. Tveganja te vrste so največja v javnih oblakih, ki so zaradi manjših stroškov najpogosteje uporabljeni:

- **Prepustitev nadzora:** Uporabnik prepusti nadzor nad svojimi podatki ponudniku storitev. Podatke ponudnik hrani in obdeluje na računalnikih, nad katerimi uporabnik nima nadzora. Zaupanje med uporabnikom in ponudnikom storitev mora biti zato veliko.
- **Nedovoljena uporaba:** Obstaja možnost, da bodo uporabnikovi podatki uporabljeni v nedogovorjene namene, ali nedovoljena prodaja osebnih podatkov oglaševalcem ali pa prodaja poslovnih podatkov podjetja konkurenci.
- **Razpršenost pri shranjevanju podatkov:** Ponudnik storitev v oblaku lahko podatke in njihove varnostne kopije shranjuje na različnih strežnikih na različnih lokacijah, tudi v različnih državah. Zakonodaja posameznih držav glede varstva osebnih podatkov se lahko močno razlikuje med seboj.
- **Dinamično delovanje:** Poleg razpršenosti lahko neskladje z zakonodajo glede uporabe osebnih podatkov povzroči že sama dinamična narava storitev v oblaku. Oblak se dinamično prilagaja nalogam in obremenitvam, ponudnik storitev lahko uporablja tudi najeto arhitekturo. Pregled in nadzor nad varstvom podatkov lahko postaneta v takem okolju težavna.

2.3.2 Tveganja, povezana z varnostjo podatkov

Če lahko privzamemo, da je v zasebnem oblaku za varnost poskrbljeno oz. je zanjo odgovorno podjetje – lastnik oblaka, to v javnih in hibridnih oblakih ne velja. Tveganja, povezana z varnostjo so:

- **Dostop:** Po eni strani gre za fizični dostop (uporaba oblaka lahko poveča tveganja, povezana z dostopom do zaupnih informacij). Do podatkov lahko dostopajo npr. vlade tujih držav, pri čemer lahko imajo (glede na zakonodajo) do tega povsem zakonito pravico in jim morda lastnika podatkov

o dostopu sploh ni treba obvestiti. Po drugi strani lahko pride do nepooblaščenega dostopa, kot se pojavlja na strežnikih, priključenih v internet. V primeru oblaka se možnost poveča, saj za strojno opremo in varnost skrbi ponudnik storitev v oblaku, tako da je varnost odvisna od internega znanja ponudnika in njegovih zaposlenih.

- **Nadzor nad življenjsko dobo podatkov:** Eden večjih izzivov uporabe storitev v oblaku je, kako poskrbeti, da ima uporabnik nadzor nad svojimi podatki — še posebej gre tu za brisanje. Težava je še toliko večja zaradi delovanja oblaka: varnostne kopije, replikacija podatkov ipd. Kako zagotoviti, da bodo brisani podatki res izbrisani? Med uporabnikom storitev in ponudnikom storitev mora obstajati neka mera zaupanja.

- **Varnostne kopije:** V oblaku se lahko zaradi zagotavljanja neprekinjenega dostopa varnostne kopije izvajajo samodejno, brez vednosti uporabnika. To lahko predstavlja varnostno tveganje, saj bi lahko npr. zaradi napake v avtomatizmu prišlo do nepopolnega kreiranja varnostne kopije, lahko pa bi postopek tudi končali s spremenjenimi podatki brez ustrezne varnostne kopije.

- **Pomanjkanje standardizacije:** Storitvam v oblaku manjka standardizacija – ni standardov za interoperabilnost, standardnega podatkovnega formata... To lahko povzroči težave pri menjavi ponudnika storitev v oblaku, ki lahko uporablja drugačno rešitev kot prvotni ponudnik.

- **Revizija:** Spet je ponudnik storitev tisti, ki mora omogočiti zunanjo revizijo svojega delovanja. Ker je okolje storitev v oblaku kompleksno, mora ponudnik poskrbeti za interni pregled vseh transakcij, ki potekajo v oblaku, tako da s tem zagotovi uporabniku, da njegovi podatki niso bili spremenjeni brez vednosti in da je vsaka transakcija zabeležena.

2.4 Virtualizacija in tehnologija vsebnikov v storitvah v oblaku

Ni težko videti, da uporaba navideznih strojev za strežbo oblačnih virov omogoča realizacijo različnih karakteristik storitev v oblaku, npr. storitev na zahtevo in elastičnost. Ali če pogledamo storitveni model – storitve vrste IaaS si težko predstavljamo brez uporabe virtualizacije.
Pri storitvah v oblaku se uporabljajo različne funkcionalnosti programske opreme za virtualizacijo: navidezne stroje lahko sorazmerno hitro prižigamo in ugašamo, začasno ustavljamo, kloniramo, migriramo in se s tem dinamično prilagajamo uporabnikom in njihovim zahtevam. Navidezni stroji so vključeni v virtualna omrežja, tako da je za uporabnika dostop do posameznega vira enak, ne glede na lokacijo. Poleg tega so navidezni stroji med seboj izolirani, uporabniki ne vidijo drugih uporabnikov, virtualizacijska plast pa uporabnikom storitev skrije tudi to, da si gostiteljeve vire delijo.

Virtualizacija je, če že ne moremo reči omogočila, vsaj olajšala in pospešila razvoj storitev v oblaku. Hkrati razširjenost in popularnost teh storitev spodbujata razvijalce programske opreme za virtualizacijo k nenehnim izboljšavam in nadgradnjam lastnih produktov, kot tudi proizvajalce procesorjev k izboljšanju strojne podpore virtualizaciji in razvoju novih funkcionalnosti, povezanih z virtualizacijo (kot je npr. podpora virtualizaciji V/I naprav). Napisano velja seveda tudi za tehnologijo vsebnikov, s katero lahko pri storitvah v oblaku na tehnološko drugačen način dosežemo enak rezultat kot z virtualizacijo.

Poglavje 3

Virtualizacija

Virtualizacija [5, 24, 25] je programska rešitev, ki na enem računalniku omogoča sočasno poganjanje več operacijskih sistemov. Ti gostujoči operacijski sistemi tečejo vsak v svojem izoliranem okolju – navideznem stroju (*virtual machine*). Navidezne stroje ustvarja, ugaša in nadzoruje nadzornik navideznih strojev (*virtual machine monitor*, *VMM*) ali hipervizor (*hypervisor*), ki deluje na fizični strojni opremi. Navidezni stroj je kopija pravega računalnika z navidezno strojno opremo in gostujoči operacijski sistem v njem deluje popolnoma enako, kot bi na realni strojni opremi.

Sočasno gostovanje več operacijskih sistemov na enem računalniku nam omogočata emulacija in virtualizacija. Emulacija omogoča tudi gostovanje operacijskih sistemov druge arhitekture, kot jo ima gostujoči računalnik (gostitelj). Emulator mora v takem primeru poskrbeti za popolno emulacijo ukazov aplikacij in operacijskega sistema, tako da sproti prevaja vsak ukaz gosta v arhitekturo gostitelja. Če imata gost in gostitelj isto arhitekturo, bo emulacija v primerjavi z virtualizacijo imela precej slabše performančne zmogljivosti.

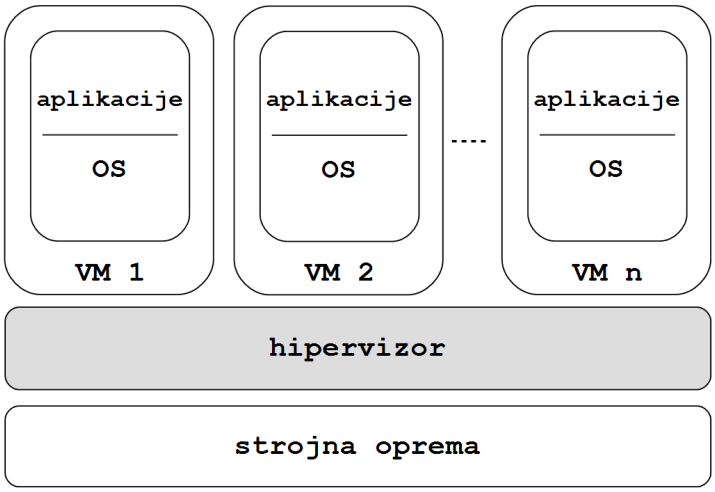
Virtualizacija deluje drugače – čim več ukazov poskuša izvršiti nespremenjenih. Hipervizor mora poskrbeti za prevajanje privilegiranih ukazov, neprivilegiranih pa ne spreminja in jih izvršuje neposredno na CPE. Privilegirani ukazi so ukazi, ki vplivajo na stanje strojne opreme, izvaja jih jedro operacijskega sistema, neprivilegirani pa so ukazi aplikacij. Performančne zmogljivosti virtualiziranega sistema pri izvajanju večinoma aplikacij so primerljive z realnim sistemom.

3.1 Zgodovina virtualizacije

Začetek virtualizacije [4, 21] sega v 60. leta 20. stoletja, ko so v IBM-ovih razvojnih laboratorijih razvili operacijski sistem CP/CMS. CP je bil hipervizor z možnostjo polne virtualizacije strojne opreme in je lahko ustvaril več med seboj neodvisnih virtualnih strojev, na katerih je tekel CMS kot operacijski sistem. Ta tehnologija je tekla na IBM-ovih sistemih System 360 in System 370. Sprva je bila virtualizacija mišljena le kot raziskovalno orodje, leta 1972 pa jo je IBM dal tudi na trg pod imenom VM/370 (v kombinaciji s prvimi strežniki z navideznim pomnilnikom S/370).
Virtualizacija je doživela manjši zaton, ko se je v 70. in 80. letih 20. stoletja hitro razvijala računalniška oprema, kar je povzročilo premik iz velikih dragih osrednjih računalnikov (*mainframe*) na manjše cenejše in učinkovitejše strežnike. Kasneje je sledil pojav t.i. osebnih računalnikov (PC), vse to pa je prineslo tudi nižje cene opreme. Cenovno ugodnejša oprema je omogočila razvoj distribuiranih sistemov (*cluster*, *server farm*), to je med seboj povezanih delovnih postaj, katerih skupna računska moč je presegala zmožnosti nekdanjih osrednjih računalnikov.
Težava z distribuiranimi sistemi je bila, da je taka postavitev kljub uporabi standardiziranih komponent zahtevala specializirano strojno opremo, ki je omogočala delovanje. To je pomenilo večje stroške vzpostavitve in vzdrževanja takega sistema, hkrati pa taki sistemi niso bili popolnoma izkoriščeni.
V poznih 90. letih 20. stoletja se je razvila virtualizacija na sistemih z arhitekturo x86, ki so bili sorazmerno poceni in na voljo v velikih količinah. Razširjenost virtualizacije na procesorjih x86 se je nato še povečala v letih 2005 in 2006, ko sta Intel in AMD na trg dala na voljo prve procesorje s podporo virtualizaciji (tehnologiji VT-x in AMD-V).

3.2 Virtualizacija arhitekture x86

Za arhitekturo x86 oz. drugače rečeno osebne računalnike je danes na voljo kar nekaj virtualizacijskih rešitev. Če naštejemo le nekaj najpopularnejših: produkti podjetja VMWare, Oracle Virtualbox, Microsoft Hyper-V, Xen, Apple Parallels,... Na voljo so za različne operacijske sisteme in za delovanje uporabljajo eno ali več načinov virtualizacije – v zadnjem času pravzaprav vse rešitve uporabljajo (tudi)



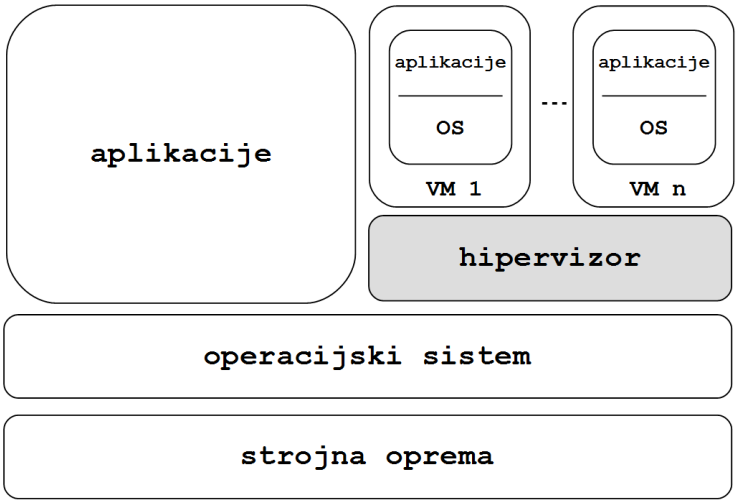
Slika 3.1: Hipervizor tipa I;
VM 1 – VM n so navidezni stroji

strojno podprto virtualizacijo. V naslednjih poglavjih si bomo pogledali, zakaj je bila virtualizacija arhitekture x86 na začetku trd oreh. Nadalje bomo predstavili delovanje vseh treh načinov virtualizacije. Za pregled delovanja polne virtualizacije z uporabo binarne translacije bomo uporabili primer rešitve podjetja VMWare, za katero najdemo precej literature [2, 7, 23], za prikaz delovanja paravirtualizacije pa primer rešitve, kot jo uporablja Xen [3, 22]. Zanimajo nas osnovni načini delovanja, predvsem da bomo razumeli, na katerih področjih in zakaj prihaja do stroškov režije (overhead) pri uporabi virtualizacije. Preveč podrobno tega ne bomo obravnavali, saj to ni v obsegu tega dela.
Tu moramo omeniti, da v literaturi najdemo različne uporabe pojmov nadzornik navideznih strojev in hipervizor. Glede na način izvedbe lahko hipervizorje delimo na dve vrsti:

1. Tip I – hipervizor teče neposredno na strojni opremi (*bare metal architecture*).

2. Tip II – hipervizor teče kot aplikacija znotraj gostiteljevega operacijskega sistema (*hosted architecture*).

V nekateri literaturi med pojmoma nadzornik navideznih strojev in hipervizor ne razlikujejo. Drugje, predvsem razvijalci določenih rešitev (VMWare, Xen), pa

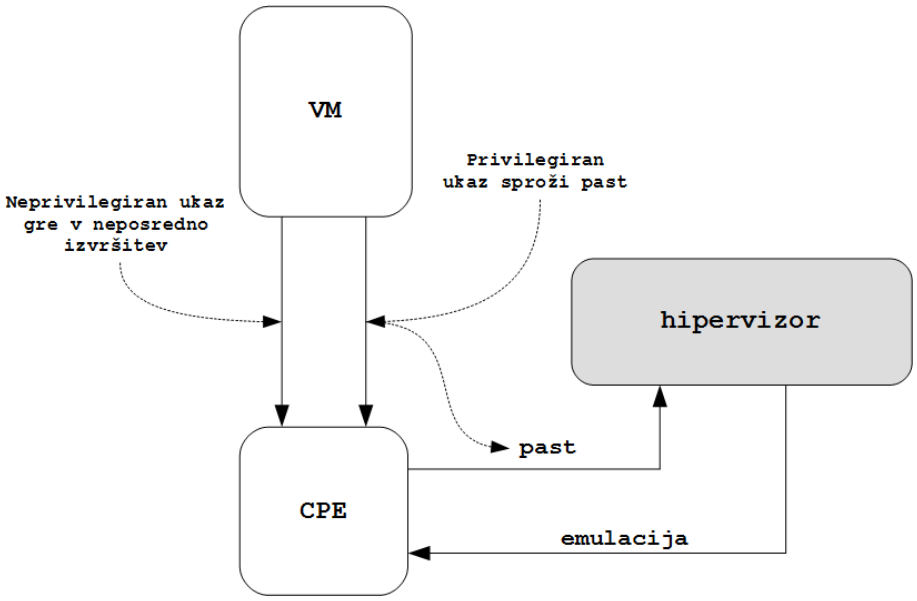


Slika 3.2: Hipervizor tipa II

hipervizor tipa I imenujejo preprosto hipervizor, hipervizor tipa II pa nadzornik navideznih strojev. Velja omeniti, da ne moremo postaviti stroge ločnice med obema vrstama – če pogledamo KVM, ki je modul Linuxovega jedra, bi ga lahko šteli med enega ali drugega.
V našem delu bomo pojma nadzornik navideznih strojev in hipervizor enačili, uporabljali pa bomo pojem hipervizor.

3.2.1 Problematika virtualizacije arhitekture x86

Leta 1998, ko smo dobili prvi hipervizor za x86 (VMWare Workstation), arhitektura x86 ni imela strojno podprte virtualizacije. Splošno mnenje je bilo, da je arhitekturo x86 nemogoče ali vsaj nepraktično virtualizirati. Na to mišljenje je vplivalo kar nekaj dejstev. Prvo je bilo že sama zasnova arhitekture – njeni začetki segajo v 70. leta 20. stoletja, ko je virtualizacija vladala na velikih osrednjih računalnikih (*mainframe*), tako da to ni bil eden izmed željenih ciljev razvoja. Kot drugi razlog je bila največkrat navedena kompleksnost. Gre namreč za arhitekturo vrste CISC (*Complex Instruction Set Computer*), ki že sama posebi nakazuje kompleksnost pri poskusu implementacije virtualizacije. Do danes je arhitektura dobila precej nadgradenj: zaščiten način delovanja (*protected mode*) v 286 CPE v 80. letih 20. stoletja, 32-bitna razširitev in delna podpora virtualizaciji realnega



Slika 3.3: Virtualizacija na način "ujemi in emuliraj" (*trap-and-emulate*)

načina delovanja (*real mode*) v 386 CPE v 90. letih 20. stoletja, 64-bitna razširitev v letu 2003, kasneje še večjedrna zasnova. Kljub vsem tem velikim spremembam oz. razširitvam je v arhitekturi ostala kompatibilnost za nazaj oz. podpora zgodovini (*legacy*), kar pomeni, da lahko tudi na najnovejših x86 CPE poganjamo stare operacijske sisteme in aplikacije. Ta pomembna značilnost arhitekture je največ prispevala k njeni popularnosti, po drugi strani pa s tem otežuje implementacijo virtualizacije, ker je treba za popolno emulacijo arhitekture potrebno podpreti veliko načinov delovanja in razširitev.

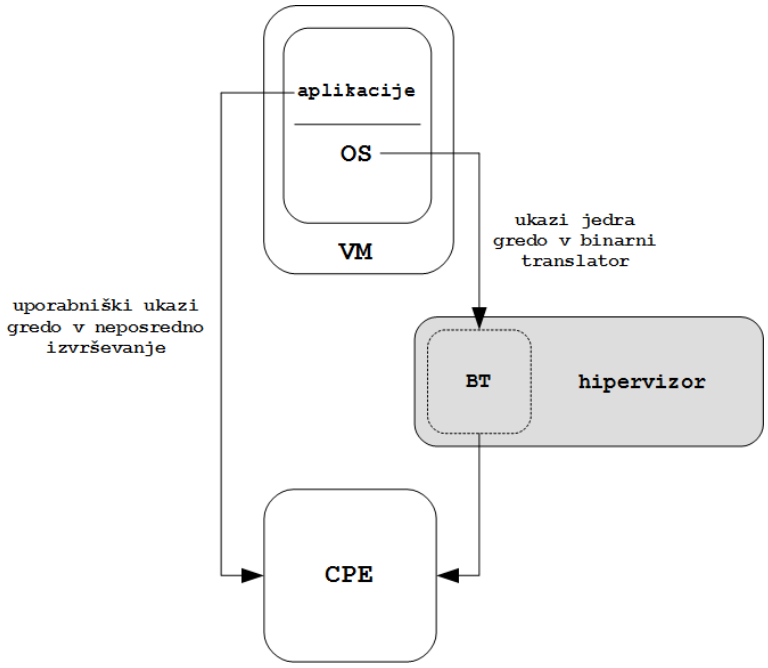
Kljub vsemu pa zgoraj navedeno samo po sebi ne izključuje možnosti virtualizacije. Glavni razlog za mnenje o nezmožnosti virtualizacije arhitekture x86 je drugje, in sicer gre za nezmožnost CPE, da sproži ustrezne pasti (*traps*) oz. napake (*faults*), kot jim pravi terminologija arhitekture x86, da bi lahko uporabili način "ujemi in emuliraj" (*trap-and-emulate*).

"Ujemi in emuliraj" (slika 3.3) je pri mnogih drugih arhitekturah dokaj tipičen način virtualizacije. Pri tem načinu se gostova koda izvaja neposredno na CPE, pri čemer ima zmanjšano stopnjo privilegiranosti. Če gost želi izvesti ukaz, ki bo bral ali spreminjal privilegirana stanja, CPE sproži past in prepusti nadzor hiper-

vizorju. Le-ta izvede emulacijo ukaza s pomočjo interpreterja, nato pa nadaljuje z neposrednim izvajanjem (*direct execution*) naslednjih gostovih ukazov.
Glede privilegiranosti izvajanja ukazov pozna arhitektura x86 štiri nivoje privilegiranosti, in sicer so to nivoji (*rings*) 0–3, pri čemer je nivo 0 najbolj privilegiran, nivo 3 pa najmanj. Večina operacijskih sistemov uporablja le dva nivoja, nivo 0 za izvrševanje ukazov operacijskega sistema (t.i. način delovanja jedra – *kernel mode*) in nivo 3 za aplikacije (uporabniški način – *user mode*). Nivoja 1 in 2 tako tipično nista uporabljena. Pri virtualizaciji imamo dva načina delovanja, gostiteljev (*host*) in gostujoči (*guest*), pri čemer je gostiteljev način "normalen" način delovanja (ni delujočega hipervizorja). V gostujočem načinu pa deluje navidezni stroj (gost), ki ga krmili in nadzoruje hipervizor. Dokler gost izvaja le ukaze aplikacij (nivo 3), ni težav, hipervizor poskrbi za pravilno uporabo pomnilnika in izvaja ukaze neposredno na CPE.
Kako pa je z ukazi operacijskega sistema oz. jedra? Ker pri virtualizaciji med strojno opremo in operacijski sistem vrinemo virtualizacijski nivo — hipervizor, gostujoči operacijski sistem ne teče več na najbolj privilegiranem nivoju. Tu pri arhitekturi x86 predstavljajo največji problem t.i. na virtualizacijo občutljivi ukazi (*virtualization sensitive instructions*) zaradi katerih je izvedbo "ujemi in emuliraj" nemogoče realizirati. Dober primer je ukaz *popf*, ki naloži zastavice (*flags*) iz sklada (*stack*) v namenski register (register *%eflags*). Če ukaz izvede jedro operacijskega sistema, ki je v privilegiranem načinu, bo v register naložil vse zastavice (vključno s sistemskimi). Če pa ga izvede program, ki teče v uporabniškem načinu, se ukaz tudi izvede, le da sistemske zastavice pusti nedotaknjene – napake pa ne sproži. V virtualiziranem sistemu pa gostovo jedro ne teče v privilegiranem načinu, zato izvajanje ukaza *popf* v takem primeru ne bi imelo pričakovanih rezultatov, hipervizor pa zaradi nesprožitve pasti ne bi dobil nobene informacije, da je šlo kaj narobe oz. da rezultat izvršitve ukaza ni tak, kot je pričakovano.

3.2.2 Polna virtualizacija z uporabo binarne translacije

Prvo rešitev za virtualizacijo arhitekture x86, VMware Workstation, je leta 1999 izdalo podjetje VMware, pri čemer so za izvedbo rešitve uporabili binarno tran-



Slika 3.4: Polna virtualizacija z uporabo binarne translacije (BT)

slacijo (*binary translation*)[2, 7].

Binarna translacija (tudi dinamična translacija) je programska tehnika za virtualizacijo, s katero hipervizor pridobi sposobnost prestrezanja za virtualizacijo občutljivih ukazov brez zanašanja na pasti oz. napake procesorja. Binarni prevajalnik kot vhodne podatke sprejema polno množico ukazov arhitekture x86, vrača pa podmnožico ukazov, ki so neprivilegirani. Prevajalnik deluje dinamično, med izvajanjem, ukaze pa prevede tik preden gredo v izvrševanje (kar reši problem ločevanja ukazov od podatkov). Ko hipervizor pošlje skozi prevajalnik neki ukaz, ga ta prevede v zaporedje ukazov, s katerim potem hipervizor emulira izvajanje privilegiranega ukaza v navideznem stroju in uveljavi spremembe na navidezni strojni opremi. Za boljšo učinkovitost hipervizor translacijo uporablja le pri ukazih, ki jih izvaja jedro operacijskega sistema gosta (*kernel*), ki teče v privilegiranem načinu delovanja. Prevajalnik prevaja le privilegirane in na virtualizacijo občutljive ukaze, vse preostale lahko prevede brez sprememb ali z malo spremembami. Za ukaze, ki se izvršujejo v uporabniškem neprivilegiranem načinu delovanja (*user mode*), pa lahko uporabi neposredno izvrševanje.

Ukaze prevajalnik prevede, ko jih gost želi prvič izvršiti, prevedene ukaze pa hrani

v posebnem medpomnilniku. Če pride do izvršitve istih ukazov v prihodnosti, hipervizor uporabi prevod iz medpomnilnika. S tem se zmanjšajo stroški režije, ki nastanejo pri prevajanju.

Virtualizacija navideznega pomnilnika

Navidezni pomnilnik je mehanizem za preslikavo več navideznih naslovnih prostorov (ponavadi ima vsak proces svojega) v manjši fizični naslovni prostor. Arhitektura x86 je navidezni pomnilnik dobila s procesorji 80386, ki so dobili za to namenjeno enoto za upravljanje pomnilnika (*memory managment unit*), v nadaljevanju MMU. Za preslikavo arhitektura uporablja hierarhične tabele strani (*page tables*), ki so shranjene v pomnilniku. Drevo teh tabel, ki ga določa korenska tabela, tako določa celotno preslikavo iz navideznega v fizični prostor. Enota MMU uporablja dve strukturi: sprehajalca po tabelah strani (*table walker*) in poseben medpomnilnik (*translation lookaside buffer*), v nadaljevanju TLB. Ko želi neki ukaz dostopati do nekega navideznega naslova, mora sprehajalec preiskati tabele strani, da dobi ustrezen fizični naslov, najdeno preslikavo pa shrani v TLB za pospešitev iskanja te preslikave v prihodnosti.
Ker gost pričakuje poln naslovni prostor (npr. v primeru 32-bitnega gosta gre za 4 GiB), mora hipervizor poskrbeti tudi za virtualizacijo navideznega pomnilnika. Za to nalogo ima svoj programski MMU, s katerim mora poskrbeti, da gost ne dostopa do pomnilnika, ki mu ne pripada (do gostiteljevega ali hipervizorjevega ter pomnilnika drugih gostov), in da je gostov navidezni prostor ustrezno preslikan.
Da omogoči gostu hiter dostop do pomnilnika, hipervizor uporablja hardverski TLB in dodatne "senčne" tabele strani (*shadow page tables*). Za vsako tabelo strani, ki jo uporablja gost, mora hipervizor zgraditi senčno tabelo, v kateri hrani združeno preslikavo iz gostovega navideznega v gostiteljev fizični prostor. Tako lahko hipervizor hardversko MMU neposredno usmeri na prave naslove in zagotovi hiter dostop do pomnilnika, hkrati pa ta način omeji dostop do pomnilnika le na dovoljene naslove. Podrobnosti delovanja so zunaj obsega tega dela, zato jih ne bomo obravnavali. Vendar lahko povemo, da zgrešitve pri gostovih dostopih do pomnilnika stanejo dosti več kot pri običajne zgrešitve v TLB, saj mora sprehajalec pregledati dodatne senčne tabele, nekaj pa prispeva tudi vzdrževanje teh tabel (tako s stališča porabe prostora kot procesorskega časa).

Uporaba segmentacije

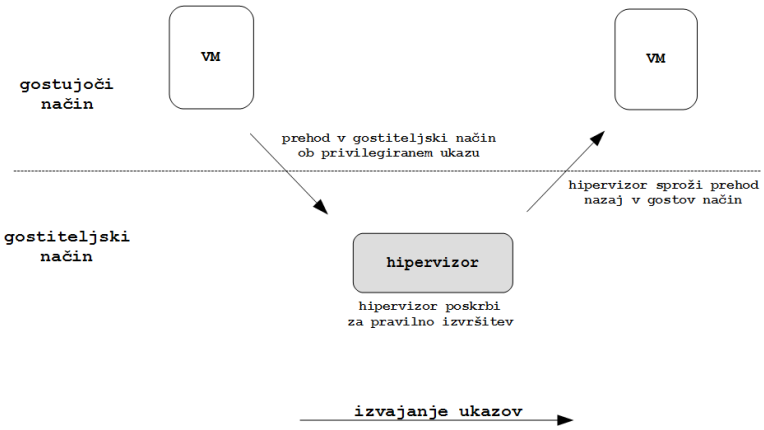
Omeniti velja še naslednji problem v povezavi z gostovim pričakovanjem, da ima na voljo poln navidezni naslovni prostor. Hipervizor mora poskrbeti tako za gostova pričakovanja kot tudi za svoje pomnilniške potrebe. Potrebuje namreč nekaj prostora za medpomnilnik za translacijo in druge pomožne strukture, kar pomeni, da mora omejiti gostov dostop do dela naslovnega prostora, ki je na voljo izključno hipervizorju. Rešitev te omejitve za neprivilegirane ukaze je na ravni tabel strani, za ukaze gostovega jedra pa uporaba segmentacije (*segmentation*), mehanizma, ki je del arhitekture x86 že od procesorjev 80286 naprej.

Virtualizacija 64-bitne arhitekture x86

Leta 2003 je AMD izdal 64-bitni procesor z razširitvijo arhitekture x86, poimenovano AMD64 oz. x86-64, tudi x64. Dodali so nov način delovanja, 64-bitni način oziroma t.i. *long mode*, hkrati pa ohranili vse predhodne načine delovanja (16-bitni, 32-bitni) in tako ohranili kompatibilnost za nazaj. Druge spremembe so bile še na ravni registrov, nekatere so razširili na 64 bitov, dodali nekaj novih splošnih, hkrati pa spremenili tudi določene registre v uporabi pri segmentaciji. Ta sprememba je onemogočila uporabo segmentacije v namene ločevanja naslovnega prostora gosta in hipervizorja pri binarni traslanciji na učinkovit način. AMD je kasneje to popravil in vrnil manjkajočo funkcionalnost, Intel pa v svoji izvedbi 64-bitne razširitve (IA-32 oz. EM64T) tega ni storil, zato z Intelovimi procesorji učinkovita virtualizacija 64-bitne arhitekture brez strojne podpore (VT-x) ni možna, medtem ko je z AMD-jevimi procesorji mogoča.

3.2.3 Strojno podprta virtualizacija

Žal na področju strojne podpore virtualizaciji arhitekture x86 ni prišlo do standardizacije, tako da sta leta 2005 Intel in AMD predstavila vsak svojo rešitev, ki omogoča virtualizacijo arhitekture x86 na način “ujemi in emuliraj”. Razširitvi (Intel) VT-x in AMD-V se razlikujeta v podrobnostih izvedbe, delujeta pa podobno in omogočata virtualizacijo brez uporabe programskih rešitev, kot je binarna translacija.



Slika 3.5: Preklapljanje gost – hipervizor pri virtualizaciji s strojno podporo

Prva generacija

VT-x in AMD-V štejemo v *prvo generacijo* strojne podpore virtualizaciji. Pri obeh igra glavno vlogo posebna nadzorna struktura v pomnilniku (*virtual machine control block* – VMCB pri AMD-V, *virtual machine control structure* – VMCS pri VT-x) za vsak navidezni stroj. Ta nadzorna struktura hrani stanje navidezne CPE posameznega navideznega stroja in njegovo nadzorno stanje. Razširitvi dodajata tudi nov, manj privilegiran način delovanja – gostujoči način (*guest mode* pri AMD-V, *VMX non-root operation* pri VT-x), v katerem izvaja gost svoje ukaze. Na drugi strani deluje hipervizor v gostiteljskem načinu (*host mode* pri AMD-V, *VMX root operation* pri VT-x). Hipervizor z izvršitvijo posebnega ukaza prepusti nadzor gostu, ki teče v gostujočem načinu – takrat shrani trenutno stanje CPE in naloži gostovo stanje iz VMCB/VMCS. Ko gost izvede privilegiran ukaz (ali če pride do izjeme), sproži s tem ukaz prehoda v gostiteljski način, pri čemer se shrani trenutno stanje gosta v VMCB/VMCS. Hipervizor na podlagi teh podatkov ustrezno poskrbi za pravilno izvršitev, nato pa spet sproži prehod v gostov način. Prehajanje iz gostujočega v gostov način (slika 3.5) in obratno potrebuje določeno število ciklov procesorja (vsaka kasnejša generacija CPE je to število zmanjševala), kar lahko pri preprostejših operacijah prinese sorazmerno visoke stroške režije. Prva generacija ni imela podpore za virtualizacijo enote za upravljanje pomnilnika, kar je pomenilo, da je moral za programsko virtualno MMU (npr. z uporabo senčnih

strani) poskrbeti hipervizor, kar pomeni še večje število prehodov iz gostujočega v gostov način in obratno. Posledica tega je bila, da prva generacija strojne podpore v določenih primerih ni dosegala performančnih zmogljivosti polne virtualizacije z uporabo binarne translacije.

Druga generacija

Druga generacija strojne podpore virtualizaciji je dodala podporo strojni virtualizaciji MMU. Kot pri prvi generaciji sta Intel in AMD ponovno predstavila vsak svojo razširitev, ki delujeta na enak način - Intel EPT (*Extended Page Tables*) in AMD RVI (*Rapid Virtualization Indexing*), poznan tudi kot NPT (*Nested page tables*). Gost ima svoje tabele strani, ki preslikajo naslove iz gostovega navideznega prostora v gostov fizični prostor, razširjene tabele strani (EPT in RVI) pa preslikajo naslove iz gostovega fizičnega prostora v realni fizični prostor. Gost lahko sam spreminja svoje tabele strani brez intervencije hipervizorja, tako da prehodi iz gostovega v gostujoči način kot pri prvi generaciji niso več potrebni. Dodatna sprememba so oznake na vnosih TLB-ja, ki označujejo, kateri navidezni CPE pripada posamezen vnos. Tako pri prehodu iz gostovega v gostujoči način praznjenje TLB-ja ni več potrebno, s čimer se izognemo tudi zmanjšanju performančnih zmogljivosti, ki bi se sicer pojavilo pri praznenju.
EPT in RVI delujeta le v povezavi z VT-x oziroma AMD-V, tako da ju ni možno kombinirati s polno virtualizacijo. Čeprav uporaba EPT in RVI pomeni višjo ceno pri zgrešitvah pri gostih (zaradi dvostopenjske translacije naslova), v splošnem pomeni boljše performančne zmogljivosti v primerjavi z uporabo programske MMU.

Druge razširitve

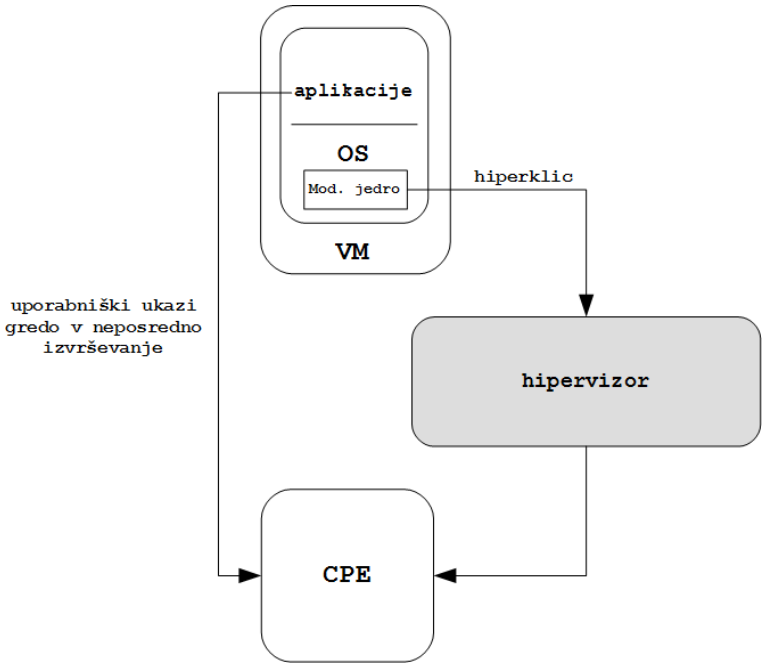
Poleg izboljšanja prehoda med gostujočim in gostiteljskim načinom, Intel in AMD razvijata še druge razširitve za virtualizacijo. Ena izmed razširitev je namenjena podpori virtualizaciji V/I naprav, Intel VT-d in AMD IOMMU. Razširitev omogoča neposredno dodelitev V/I naprave izbranemu virtualnemu stroju – v takem primeru možnost deljene uporabe te naprave ni več možna, kar ni nujno težava.

3.2.4 Paravirtualizacija

Prednost polne in strojno podprte virtualizacije je v popolni abstrakciji strojne opreme, tako da je gostujoči operacijski sistem izoliran od hipervizorja, gostiteljevega sistema in drugih morebitnih navideznih strojev. Prednost, ki jo dobimo z možnostjo namestitve katerega koli nemodificiranega operacijskega sistema, ustvari določene stroške režije, ki poslabšajo performančne zmogljivosti virtualiziranega sistema v primerjavi s fizičnim.
Paravirtualizacija je rešitev z drugačno idejo. Pri tej obliki virtualizacije operacijski sistem gosta oz. njegovo jedro "ve", da teče v navideznem stroju. Še več, pri virtualizaciji sodeluje s hipervizorjem. S tem se izognemo stroškom, kot jih imata polna in strojno podprta virtualizacija, slabost pa je, da smo omejeni le na operacijske sisteme, katerih jedro podpira paravirtualizacijo. Pri tem je treba omeniti, da gostovih aplikacij ni treba modificirati.
Spodaj opisani načini delovanja paravirtualizacije se nanašajo na rešitev, kot je izvedena s hipervizorjem Xen [3, 22].

Izvedba privilegiranih ukazov

Enako kot pri polni virtualizaciji se tudi pri paravirtualizaciji srečujemo s problemom privilegiranih in na virtualizacijo občutljivih ukazov. Gostov operacijski sistem ima modificirano jedro tako, da ukaze izvršuje na nivoju privilegiranosti 1 (*ring* 1). S tem je poskrbljeno, da gost ne more neposredno izvrševati privilegiranih ukazov, hkrati pa gostov operacijski sistem ostaja zaščiten pred gostovimi aplikacijami, ki tečejo na nivoju 3. Privilegirane operacije so paravirtualizirane – to pomeni, da jih mora validirati in izvršiti hipervizor, ki teče na nivoju 0. Zahteve za izvršitev privilegiranih operacij gredo v hipervizor preko t.i. hiperklicev (*hypercall*), katerih ustreznica v navadnem operacijskem sistemu so sistemski klici (slika 3.6).
Za izjeme, kot so napake in pasti, tudi poskrbi hipervizor, in sicer preko posebne tabele, v kateri so za vsako vrsto izjem registrirane rutine za obdelavo izjem (*exception handlers*). Ob izjemi hipervizor poskrbi za potrebne podatke o izjemi in prepusti nadzor ustrezni rutini. Izjemi, ki se pojavljata dovolj pogosto, da vplivata na zmogljivosti sistema, sta sistemski klic (*system call*) in napaka strani (*page fault*). Za pospešitev sistemskih klicev omogoča hipervizor registracijo posebne rutine za



Slika 3.6: Paravirtualizacija

obdelavo, pri kateri preusmeritev preko hipervizorja ni potrebna. Za napake strani taka pospešitev ni možna, saj lahko iz registra, ki vsebuje naslov, pri katerem je nastala napaka, bere le ukaz, ki se izvršuje na nivoju 0. Tako mora za pridobitev tega naslova poskrbeti hipervizor.

Upravljanje pomnilnika

Podobno kot pri polni virtualizaciji je tudi pri paravirtualizaciji eden zahtevnejših delov virtualizacija navideznega pomnilnika. Pri paravirtualizaciji mora za svoje tabele strani poskrbeti operacijski sistem gosta, pri čemer po kreiranju nove tabele strani le-to registrira v hipervizorju. Od takrat naprej ima le dostop za branje, vsa nadaljnja ažuriranja validira in opravi hipervizor, s čimer prepreči nezaželene spremembe v tabeli. Dodatno si hipervizor podobno kot pri polni virtualizaciji deli naslovni prostor z gostom (v primeru hipervizorja Xen gre za zgornjih 64 MiB naslovnega prostora), s čimer preprečimo zmanjšanje učinkovitosti pri prehodu iz jedra v hipervizor in nazaj.

Vhodno-izhodne naprave

Pri paravirtualizaciji V/I naprave, kot so podatkovne enote in mrežni vmesniki, niso emulirane, kot pri polni virtualizaciji. Namesto tega ima gost na voljo preproste vmesnike na voljo za prenos V/I podatkov, ki poteka preko hipervizorja, kar prinese boljše zmogljivosti v primerjavi z emuliranimi napravami.

Paravirtualizacija 64-bitne arhitekture x86

Podobno kot pri polni virtualizaciji je odstranitev možnosti segmentacije iz 64-bitnih CPE tudi povzročila težave. Kot pri polni virtualizaciji tudi paravirtualizacija uporablja segmentacijo za zaščito hipervizorja pred gostovim jedrom, saj uporabljata isti naslovni prostor zaradi učinkovitejšega delovanja. Odstranitev možnosti segmentacije pomeni, da pri paravirtualizaciji 64-bitnih gostov jedro tako kot aplikacije teče na nivoju 3 in v svojem naslovnem prostoru. To pomeni dodatne stroške režije pri izvedbi sistemskega klica gosta, ko preide nadzor najprej v hipervizor, nato pa še v jedro. V nasprotju s polno virtualizacijo so torej paravirtualizirani 64-bitni gosti možni, vendar so njihove performančne zmogljivosti v primerjavi s paravirtualiziranimi 32-bitnimi gosti slabše.

Druge paravirtualizacijske tehnike

Določene paravirtualizacijske tehnike je možno uporabiti tudi pri polni in strojno podprti virtualizaciji, pri čemer gre tu za majhne spremembe v obliki storitev in gonilnikov, ki ne zahtevajo modifikacije jedra. To so npr. paravirtualizirani gonilniki za virtualizirane V/I naprave, ki so povezane s hipervizorjem za boljše performančne zmogljivosti, ter storitve, kot je npr. VMware tools [59]. Preko slednjih dobimo dostop do hipervizorja, npr. za sinhronizacijo časa ali za izboljšanje uporabe grafičnega uporabniškega vmesnika za nadzor virtualnega stroja.

3.2.5 Gnezdena virtualizacija

Gnezdena virtualizacija [2, 5] pomeni, da znotraj navideznega stroja poganjamo hipervizor in v njem drug navidezni stroj. Primeri uporabe so v razvojnem in testnem okolju, za demonstracijo delovanja programske opreme in tudi v produkcijskih okoljih (npr. določeni OS uporabljajo virtualizacijo interno, npr. način

delovanja Windows XP v Windows 7 in kasnejših).
Možnosti uporabe gnezdene virtualizacije so različne glede na vrsto virtualizacije, v vsakem primeru pa je uporaba odvisna od podpore hipervizorja. Zavedati se moramo tudi dodatnih stroškov režije, ki jih gnezdenje prinese, npr. v primeru programsko virtualizirane MMU pri uporabi izključno polne virtualizacije imamo tako trinivojsko translacijo naslovov gnezdenega gosta.

Poglejmo si možne kombinacije gnezdenja. Zunanji hipervizor (teče na strojni opremi) označujemo s HZ, notranji ali gnezdeni hipervizor (teče v navideznem stroju) pa s HN:

- **HZ s polno virtualizacijo:** V navideznih strojih znotraj HZ ni ovir, da ne bi poganjali HN s polno ali paravirtualizacijo, saj ju HZ obravnava kot vsak drug program. HN s strojno podporo virtualizaciji je možen le, če HZ emulira ali virtualizira ustrezne razširitve, VT-x/AMD-V in EPT/RVI.
- **HZ s strojno podporo virtualizaciji:** Tu gre za enako situacijo kot pri HZ s polno virtualizacijo, torej nobenih ovir glede polne ali paravirtualizacije, pri HN s strojno podporo virtualizaciji pa mora HZ poskrbeti za ustrezne razširitve.
- **HZ s paravirtualizacijo:** Paravirtualiziran gost se "zaveda" virtualizacije in uporablja hiperklice pri komunikaciji s HZ. Za uspešno izvedbo gnezdenja virtualizacije v takem primeru mora tudi HN uporabljati hiperklice, torej gnezdena virtualizacija brez modifikacije HN ni možna.

3.3 Programska oprema za virtualizacijo arhitekture x86

3.3.1 Produkti podjetja VMware

Podjetje VMware [58] je pionir na področju rešitev virtualizacije arhitekture x86. Kot že omenjeno, je podjetje prvo dalo na trg produkt za to arhitekturo, VMWare Workstation [7]. Njihova linija produktov danes zajema namizne (Workstation za Windows in Linux, Fusion za Mac OS) in strežniške produkte (vSphere), v ponudbi pa imajo tudi hibridni oblak (vCloud Air).
Pri podjetju VMware so začeli s polno virtualizacijo z uporabo binarne translacije, njihovi produkti danes uporabljajo tudi strojno podprto virtualizacijo, za boljše performančne zmogljivosti pa imajo tudi paravirtualizirane gonilnike za V/I naprave (preko VMware Tools).
VMware izvorne kode svojih produktov ne daje na voljo, večina je plačljivih. Imajo pa tudi nekaj brezplačnih produktov, kot sta vSphere Hypervisor za strežnike z licenco, ki določa omejitve pri uporabi, in Workstation Player za namizja, ki je prost za uporabo v nekomercialne namene.

3.3.2 VirtualBox

VirtualBox [56]je odprtokodni produkt za virtualizacijo, prosto na voljo pod licenco GNU General Public License (GPL) verzije 2. Njegov trenutni lastnik je Oracle. VirtualBox je hipervizor tipa II, na voljo je za operacijske sisteme Windows, Linux, Macintosh in Solaris. VirtualBox za virtualizacijo uporablja tako polno virtualizacijo kot strojno podprto, ima pa tudi možnost uporabe paravirtualiziranega gonilnika za mrežni vmesnik. Na voljo je brezplačno, vendar za razširitve, ki jih ponuja Oracle (VirtualBox Extension Pack) velja druga licenca (prosto za domačo uporabo).
To je popularen produkt, predvsem zaradi brezplačnosti, multiplatformnosti, velikega števila funkcionalnosti in podpore skupnosti.

3.3.3 KVM

Kernel Virtual Machine – KVM[14, 40] je funkcionalnost jedra operacijskega sistema Linux, ki omogoča da Linux spremenimo v hipervizor s strojno podprto virtualizacijo. KVM razširitve CPE za virtualizacijo predstavi kot napravo */dev/kvm*, ki omogoča med drugim naslednje operacije:

- kreiranje novega navideznega stroja,
- alokacijo pomnilnika navideznemu stroju,
- poganjanje navidezne CPE,
- branje in pisanje registrov navidezne CPE,
- sprožitev prekinitve v navidezni CPE.

Poleg tega poleg obstoječih načinov delovanja (način delovanja jedra – *kernel mode* in uporabniški način – *user mode*) KVM doda nov *gostujoči način* (*guest mode*), v katerem izvaja ukaze navidezni stroj.
KVM deluje v kombinaciji s QEMU [51]. QEMU je sicer samostojni emulator za različne arhitekture, v kombinaciji s KVM pa je lahko tudi hipervizor. Med drugim QEMU poskrbi za emulacijo virtualnih naprav in za ustrezne klice v KVM, da kreira in upravlja navidezne stroje.

3.3.4 Xen

Xen (*The Xen Project Hypervisor*[60]) je trenutno edini odprtokodni hipervizor tipa I. Njegov razvoj se je začel s paravirtualizacijo [3], kasneje je pridobil še možnost strojno podprte virtualizacije. Na njegovi osnovi so zgrajene številne rešitve, npr. XenServer [61], Oracle VM [53] in Amazon EC2 [26].
Pri projektu Xen navideznim strojem pravijo domene, posebnost je nadzorna domena ali domena 0 (*Domain-0*, *Dom0*), ki se zažene prva ob zagonu Xen-a in predstavlja takorekoč gostiteljski operacijski sistem (brez nje Xen ne more delovati). Dom0 je privilegirana, ima neposreden dostop do strojne opreme in orodij za nadzor drugih navideznih strojev. V domeni 0 teče ponavadi različica operacijskega sistema Linux (ki ima že v jedro vgrajeno podporo tako za Xen kot za paravirtualizacijo), podprti pa so tudi drugi operacijski sistemi (npr. FreeBSD,

NetBSD, OpenSolaris).
Vse preostale domene (gosti) so neprivilegirane uporabniške domene (*User Domain*, *Unprivileged Domain*, *DomU*), izolirane od strojne opreme. DomU so lahko paravirtualizirane (*PV*) ali pa virtualizirane s strojno podporo (*Hardware Virtual Machine – HVM*). Xen pozna tu več različic oz. poimenovanj:

- **PV**: Paravirtualiziran gost ne potrebuje razširitev CPE za virtualizacijo, potrebuje pa jedro s podporo paravirtualizaciji in paravirtualizirane gonilnike.

- **HVM**: Strojno podprta virtualizacija potrebuje razširitve VT-x ali AMD-V. Poleg tega uporablja QEMU za emulacijo strojne opreme. Jedro s podporo PV pri HVM ni potrebno, zato lahko Xen v tem načinu gostuje tudi operacijske sisteme Windows.

- **PVHVM**: Za izboljšanje performančnih zmogljivosti lahko gosti tipa HVM uporabljajo posebne optimizirane paravirtualizirane gonilnike (*PVHVM* ali *PV-on-HVM*) in tako obidejo emulacijo diskov ter mrežnih vmesnikov. S tem pridobimo performančne zmogljivosti, podobne paravirtualiziranim gostom.

- **PVH**: PVH so paravirtualizirani gosti, ki uporabljajo paravirtualizirane gonilnike za zagon in V/I naprave, drugače pa razširitve CPE za virtualizacijo brez uporabe emulacije. Ideja gostov vrste PVH je izkoristiti najboljše lastnosti vseh načinov virtualizacije, pri čemer bi poenostavili Xen-ovo arhitekturo.
  Razvoj PVH se nadaljuje, trenutno gre v smeri ponovne izgradnje in poenostavitve arhitekture PVH (*PVHv2*) zaradi omejitev prve verzije.

Poglavje 4

Tehnologija vsebnikov

V primerjavi z virtualizacijo, pri kateri gre za poganjanje celotnega operacijskega sistema na virtualni strojni opremi, gre pri tehnologiji vsebnikov (*container technology*, *container-based virtualization*) za izolacijo aplikacij z uporabo funkcionalnosti, ki jih ponuja obstoječi operacijski sistem [1, 10] – tehnologijo vsebnikov zato imenujemo tudi virtualizacija na nivoju operacijskega sistema (*Operating System-level virtualization*), uporablja pa se tudi izraz "lahkokategorna" virtualizacija (*light-weight virtualization*).

Osnovna ideja uporabe tehnologije vsebnikov je, da v vsebnik zapakiramo izbrano programsko opremo z vsem, kar je potrebno za njeno delovanje, to je izvršilne datoteke, knjižnice, sistemska orodja in knjižnice, vse skupaj je na lastnem datotečnem sistemu, le jedro si vsebnik deli z operacijskim sistemom. Uporaba vsebnikov tako poenostavlja namestitev na produkcijska okolja, saj se izognemo nameščanju dodatnih orodij in knjižnic, morebitnim težavam z verzijami ipd., zagotovljeno pa imamo tudi, da bo v vsebnik zapakirana aplikacija ali sistem vedno delovala enako.

Zametki vsebnikov [6] sežejo že v leto 1979, ko je operacijski sistem Unix dobil funkcionalnost za spreminjanje korenskega direktorija *chroot*. Leta 1998 je v operacijskem sistemu FreeBSD izšla nadgradnja *chroot*, poimenovana *jails* [34], ki je dodala virtualiziran dostop do datotečnega sistema, uporabnikov in omrežja. Kasneje so prišle še druge rešitve, kot sta Solaris *zones* [52] in IBM AIX *workload partitions* [37]. Za operacijski sistem Linux so bile rešitve za uporabo vsebnikov na voljo že nekaj časa (Linux-VServer [46], 2003; OpenVZ [50], 2005), a so delovale

le z modificiranim jedrom. Kasneje so funkcionalnosti, ki so omogočale vsebnike, postale tudi del jedra (*control groups*, 2008), sledil pa jim je razvoj programskih rešitev za tehnologijo vsebnikov, npr. LXC (razvoj od leta 2008 naprej, verzija 1.0 februar 2014 [44]).

4.1 Tehnologija vsebnikov v operacijskem sistemu Linux

Tehnologija vsebnikov v operacijskem sistemu Linux [15] (*Linux Containers*) je virtualizacijska metoda, ki omogoča sočasno poganjanje več med seboj izoliranih sistemov Linux znotraj vsebnikov. Vsebnik je virtualno okolje s svojimi CPE, pomnilnikom, V/I napravami in omrežjem. Linux-VServer in OpenVZ sta za vzpostavitev vsebnikov uporabljala vsak svoje modificirano jedro operacijskega sistema Linux. LXC in Docker, ki sta prišla kasneje, pa uporabljata funkcionalnosti[43], ki so že del sistema Linux (oz. njegovega jedra): nadzorne skupine (*cgroups*), imenski prostor (*namespace*), spreminjanje korenskega direktorija in druge, odvisno od tehnologije vsebnikov (LXC, Docker itd.).

Za boljše razumevanje, kako deluje izolacija pri vsebnikih, si poglejmo, kaj zgoraj omenjene funkcionalnosti sistema Linux sploh omogočajo:

Nadzorne skupine (*control groups*, *cgroups*): Nadzorne skupine [54] so funkcionalnost Linux-ovega jedra, ki omogoča dodelitev različnih virov (uporaba CPE, pomnilnika in omrežnih virov) ali kombinacij virov po skupinah procesov. Skupine določi, nadzoruje in spreminja uporabnik, ki lahko dinamično spreminja dostop skupin do določenih virov.

Imenski prostor (*namespace*): Sistemske vire znotraj operacijskega sistema lahko umestimo v neki imenski prostor[45], tako da bodo procesom znotraj istega imenskega prostora ti viri vidni kot njim lastni in izolirani od ostalih. Spremembe teh virov so vidne le procesom, ki pripadajo istemu imenskemu prostoru, drugim pa ne. Linux zagotavlja več imenskih prostorov, npr. za izolacijo uporabnikov in skupin, za izolacijo procesov, za izolacijo omrežnih virov, za izolacijo priklopov

datotečnega sistema (*mount point*) in še nekaj drugih.

Spreminjanje korenskega direktorija: Korenski direktorij je "najvišji" direktorij, to je direktorij, v katerem so vsi preostali direktoriji, njihovi poddirektoriji in datoteke. Korenski direktorij je za proces možno spremeniti, to lahko naredimo z ukazoma, kot sta *chroot* in *pivot_root* (omogočata spremembo korenskega direktorija oz. korenskega datotečnega sistema).

4.2 Programska oprema za tehnologijo vsebnikov

V tem poglavju bomo predstavili nekaj primerov rešitev za tehnologijo vsebnikov, ki so na voljo za operacijski sistem Linux. Podrobneje bomo predstavili LXC in Docker, ker sta med popularnejšimi rešitvami, uporabljata se v storitvah v oblaku in sta del vseh večjih distribucij (Ubuntu, Fedora). Na koncu bomo na kratko predstavili še Linux-VServer in OpenVZ, ki spadata med prve rešitve za tehnologijo vsebnikov za operacijski sistem Linux.

4.2.1 LXC

LXC [43] je uporabniški vmesnik za upravljanje funkcionalnosti Linux-ovega jedra za tehnologijo vsebnikov. Omogoča, da znotraj vsebnika poganjamo aplikacijo ali popoln sistem.
Za poganjanje aplikacije moramo vedeti, katere vire (računske, pomnilniške, mrežne...) želimo izolirati znotraj vsebnika in pripraviti ustrezno konfiguracijo. Poganjanje sistema je preprostejše, treba je izolirati vse vrste virov, za kar poskrbi LXC samodejno pri ustvaritvi vsebnika. Za dodatne nastavitve potem poskrbi uporabnik sam. Vsebnik začne teči, ko v njem poženemo neki proces in se ustavi, ko v njem zaključi izvajanje še zadnji proces.

Ustvarjanje vsebnikov

Pri ustvarjanju vsebnikov imamo dve možnosti: bodisi vsebnik najprej ustvarimo (*lxc-create*) bodisi neposredno zaženemo (*lxc-execute/lxc-start*). V prvem primeru

dobimo obstojen vsebnik (*persistent container*), ki ga lahko kadar koli uničimo (*lxc-destroy*). V drugem primeru dobimo začasen vsebnik (*volatile container*), pri katerem je uničenje samodejno, bodisi ko aplikacija znotraj vsebnika neha teči bodisi ko vsebnik ustavimo (*lxc-stop*). Pri obeh primerih lahko kot argument podamo pot do konfiguracijske datoteke, opcijski parameter pa je tudi pot do direktorija, kamor se vsebniki shranjujejo (če nam privzeta nastavitev ne ustreza).
Glede pravic pozna LXC dve vrsti vsebnikov, "neprivilegirane" in "privilegirane". Neprivilegirani so varnejši, za določanje identifikatorjev uporabnikov in skupin znotraj vsebnikov (*user id, uid*, *group id, gid*) uporabljajo preslikavo. Tako ima korenski uporabnik (*root*) znotraj vsebnika uid 0, zunaj vsebnika pa neki drug uid z omejenimi pravicami. Če bi se izvedel napad na vsebnik, zaradi katerega bi napadalcu uspel izhod iz izolacije vsebnika, bi napadalec končal kot uporabnik z omejenimi pravicami. Neprivilegirani vsebniki imajo določene omejitve (določene operacije, kot npr. priklopi datotečnih sistemov, niso dovoljene), ki pa se razlikujejo, če jih ustvarimo kot navaden uporabnik ali kot korenski uporabnik. Privilegirane vsebnike ustvarimo kot korenski uporabnik brez preslikav, tako da ima korenski uporabnik znotraj vsebnika enake pravice kot korenski uporabnik gostiteljskega sistema. Vsi procesi znotraj vsebnika tečejo s pravicami korenskega uporabnika, zato je skrb za varnost še posebej pomembna. Prednost privilegiranih vsebnikov pa je, da nimajo omejitev neprivilegiranih.

Poleg ukazov za kreiranje, zagon, ustavitev in uničenje vsebnikov, ima LXC še druga orodja, npr. za pregled stanja vsebnikov, "zamrznitev" in "odmrznitev" stanja vsebnika, spreminjanje nastavitev nadzornih skupin že delujočega vsebnika.

Če se omejimo na poganjanje sistema znotraj vsebnika, potem skozi oči uporabnika LXC deluje zelo podobno kot virtualizacijske rešitve: uporabnik ustvari vsebnik na podlagi neke predloge (*template*), vsebniku določi število CPE in velikost pomnilnika ter ga umesti v omrežje. Predloga je ustreznica sliki (*image*) operacijskega sistema, LXC pri ustvarjanju vsebnika ponudi več že pripravljenih predlog, ki jih prenese s spleta, in so ustreznice popularnejšim operacijskim sistemom: Debian, Ubuntu, Fedora, openSUSE, Arch Linux in drugi. Za vsako distribucijo so na voljo predloge za različne verzije in arhitekture (x86, x86-64, ARM idr). Poleg

omenjenih predlog LXC uporabniku omogoča tudi ustvarjanje svojih.

Konfiguracija vsebnikov

Vsak vsebnik ima svojo konfiguracijsko datoteko, v kateri določimo, kateri viri so na voljo vsebniku, ter druge nastavitve. Med drugim vnosi v datoteki določajo naslednje stvari:

- Signale za ustavitev (*shutdown*), ponovni zagon (*reboot*) in prisilno ustavitev vsebnika.
- Inicializacijski ukaz (*init command*), ki se zažene pri vsebnikih s polnim sistemom.
- Omrežne nastavitve: vsebnik je lahko brez omrežja, priključen preko navideznega vmesnika (*virtual ethernet, veth*) na gostiteljev most, priključen preko posebnega vmesnika v neko navidezno omrežje (*virtual lan, vlan*) ali neposredno uporablja obstoječ fizični vmesnik.
- Nastavitve korenskega direktorija vsebnika ter priklopov datotečnih sistemov in pravic dostopov.
- Nastavitve nadzornih skupin (*control groups, cgroups*) – npr. za omejitev pomnilnika ali za določitev, katera CPE je namenjena vsebniku.
- Nastavitve okoljskih spremenljivk (*environment variables*).
- Nastavitve za AppArmor, SELinux in seccomp (funkcionalnosti, povezane z izolacijo in zaščito procesov ter nadzorom dostopa), če je bila podpora dodana v LXC med prevajanjem.

4.2.2 Docker

Podobno kot LXC tudi Docker [32] uporablja funkcionalnosti Linuxovega jedra za lastno rešitev izvedbe tehnologije vsebnikov. Docker do teh funkcionalnosti dostopa bodisi neposredno preko lastne knjižnice (*libcontainer*) bodisi posredno preko drugih rešitev (npr. LXC). Načini delovanja in upravljanja vsebnikov so podobni LXC.

Delovanje omogoči prikriti process Docker Engine (pri LXC na primer, ni nekega prikritega procesa), ki poskrbi za kreiranje in upravljanje vseh vrst Dockerjevih objektov (vsebniki, predloge, omrežja in podatkovni nosilci). Z Docker Engine ne upravljamo neposredno, ampak preko odjemalca, ki komunicira z Docker Engine preko vtičnic (*socket*) ali preko vmesnika vrste RESTful [36]. Odjemalec in Docker Engine tako lahko tečeta na istem sistemu, lahko pa tudi vsak na svojem.
Omenili smo podoben način delovanja kot pri LXC. Tudi pri Docker-ju za kreiranje vsebnika potrebujemo predlogo (*Docker image*), ki jo pridobimo iz oddaljenih ali lokalnih repozitorijev (*Docker registry*), lahko jih kreiramo tudi sami (in shranimo v neki repozitorij). Predloga vsebnika vsebuje neko aplikacijo in datoteke, potrebne za njeno delovanje, ter konfiguracijske podatke. Docker ponuja več že pripravljenih predlog na svojem javnem repozitoriju, npr. predloge na podlagi operacijskih sistemov (Ubuntu, Fedora, Debian, Centos...) ali predloge z že pripravljenimi aplikacijami ali različnimi orodji (HTTP strežnik Apache, podatkovna baza MySQL, Python, PHP, Ruby...).
Podobno kot LXC pozna tudi Docker privilegirane in neprivilegirane vsebnike. Privzeto so vsebniki neprivilegirani in nimajo dostopa do naprav gostiteljskega sistema. S posebnim argumentom ob zagonu (*docker run --privileged*) pa vsebnik dobi dostop do vseh naprav in možnost spreminjanja določenih nastavitev, ki omogočajo skoraj enake pravice kot drugi procesi na gostiteljskem sistemu.

Dockerjeve predloge

Predloge so samo za branje, za njihovo delovanje Docker uporablja poseben večplastni datotečni sistem (*Union File System*). Predlogo sestavlja več plasti, začne se z osnovno plastjo (npr. osnovna slika sistema Ubuntu ali Fedora, lahko pa tudi že neka obstoječa predloga), vsaka sprememba pa doda novo plast. Ob zagonu vsebnika Docker tako predloge ne spreminja, ampak doda novo bralno/pisalno plast. To plast lahko shranimo in dobimo novo predlogo, drugače pa se ob uničenju vsebnika ta plast ne shrani, tako da se vse spremembe in podatki uničijo skupaj z vsebnikom.

Podatkovni nosilci

Za ohranitev podatkov moramo uporabiti podatkovni nosilec (*data volume*). Podatkovni nosilec je datoteka ali direktorij gostiteljskega sistema, ki je pripet neposredno v vsebnik. Branje in pisanje v nosilec sta neposredni, tako da dosega hitrosti gostiteljevega sistema (tu ni stroškov režije, povezanih z uporabo večplastnega datotečnega sistema).

Docker in omrežja

Dockerjeve vsebnike lahko vključujemo v eno ali več omrežij. Ob nastavitvi Docker pripravi nekaj privzetih možnosti (npr. most *docker0* v gostiteljskem sistemu), lahko pa omrežja definiramo po svoje (lahko celo definiramo tudi svoj omrežni vtičnik). Docker omogoča kreiranje tudi t.i. "prekrivnega omrežja" (*overlay network*), ki omogoča medsebojno povezanost več vsebnikov, ki tečejo na različnih gostiteljih.
Da sploh lahko dostopamo do aplikacij znotraj vsebnika, moramo ob njegovem kreiranju ali že v definiciji predloge izpostaviti vrata (*expose port* v Dockerjevi terminologiji), ki omogočajo dostop do neke aplikacije. Izpostavimo lahko več različnih vrat ali več zaporedij vrat, določimo lahko tudi preslikave gostiteljevih vrat na vsebnikova vrata.

Docker na operacijskih sistemih Windows in MacOS

Za razliko od LXC je Docker na voljo tudi za operacijske sisteme Windows in MacOS. Tu uporablja zanimivo rešitev. Obstajata dve možnosti, uporaba Docker toolbox ali pa uporaba Docker for Windows oz. Docker for Mac. Obe možnosti uporabljata virtualizacijo.
Docker toolbox poleg odjemalca in nekaj drugih orodij namesti tudi Oracle VirtualBox, ki ga uporablja, da znotraj navideznega stroja poganja Docker Engine. Docker for Windows in Docker for Mac sta novejši rešitvi. Prva namesto rešitve VirtualBox uporablja Microsoftov Hyper-V, druga pa Mac HyperKit, obe različici zahtevata določeno (dovolj novo) verzijo operacijskega sistema in ustrezne razširitve za strojno podporo virtualizaciji.

4.2.3 Linux-VServer in OpenVZ

Linux-VServer in OpenVZ spadata med prve rešitve za tehnologijo vsebnikov za operacijski sistem Linux. Na začetku pojem vsebnik (*container*) še ni bil v uporabi. Pri rešitvi Linux-VServer zato uporabljajo izraz "navidezni zasebni strežnik" (*Virtual Private Server*, VPS), pri rešitvi OpenVZ pa poleg VPS tudi "navidezno okolje" (*Virtual Environment*, VE) in tudi vsebnik. Obe rešitvi uporabljata modificirano Linuxovo jedro z dodanimi funkcionalnostmi, ki omogočajo izolacijo vsebnikov. Slabost uporabe modificiranih jeder je v tem, da lahko pride do zaostanka glede na razvoj Linuxovega jedra, oziroma drugače povedano, lahko nekaj časa traja, preden dobimo modificirano različico najnovejše izdaje jedra.

Razvoj Linux-VServer se je glede na domačo stran projekta malce upočasnil. Zadnja stabilna verzija popravka (*patch*) za starejšo verzijo jedra (2.6) je bila izdana leta 2008, obstaja tudi popravek za novejšo verzijo jedra (3.18) iz leta 2015, a je označen kot eksperimentalen. Zadnja verzija "uradnih" uporabniških orodij za Linux sega v leto 2004.

Razvoj OpenVZ je na drugi strani zelo aktiven in ažuren, popravki za jedra so omejeni na določene verzije operacijskih sistemov Debian in Red Hat Enterprise Linux. Tu gre za starejše verzije jeder, saj sta pri kritičnih sistemih v podjetjih stabilnost in varnost pomembnejša od novih funkcionalnosti, ki jih prinašajo najnovejše verzije jeder. OpenVZ je odprtokodna rešitev, ki si osnovo deli z rešitvijo Virtuozzo, ki je komercialne narave (plačljiva) in ponuja še dodatne funkcionalnosti v primerjavi z rešitvijo OpenVZ. Razvijalci rešitve OpenVZ sodelujejo tudi pri razvoju Linuxovega jedra, tako da rešitve, ki jih uporablja OpenVZ počasi prehajajo tudi v Linuxovo jedro (*upstream*).

Poglavje 5

Primerjava tehnologij

V tem poglavju bomo primerjali virtualizacijo in tehnologijo vsebnikov, osredotočili se bomo na funkcionalnosti, ki jih omogoča vsaka tehnologija, ugotavljali prednosti in slabosti ter morebitne tehnološke omejitve. Rešitve virtualizacije in tehnologije vsebnikov ponujajo načeloma dokaj podoben nabor funkcionalnosti, seveda ne popolnoma enakega. Razvoj teh dveh tehnologij poteka aktivno, zato lahko pričakujemo, da bodo sčasoma morebitne manjkajoče (uporabne) funkcionalnosti prešle tudi v rešitve, ki jih danes še nimajo. V nadaljevanju bomo na mestih, kjer se besedilo zadeva tako navidezne stroje kot vsebnike uporabili besedo gost za njuno nadpomenko.

5.1 Ustvarjanje in upravljanje navideznih strojev/vsebnikov

Tehnologiji sta funkcionalno zelo podobni glede upravljanja navideznih strojev in vsebnikov. Uporabnik lahko pri obeh določa osnovne konfiguracije, kot so število CPE (tudi pripenjanje na izbrano CPE), velikost pomnilnika, priključitev v eno ali več omrežij. Vendar imajo vsebniki eno večjih pomanjkljivosti glede na navidezne stroje – omejeni so na operacijski sistem in arhitekturo svojega gostitelja. Medtem ko lahko v navideznem stroju lahko teče pravzaprav kateri koli operacijski sistem za isto arhitekturo, pa pri vsebnikih v sistemu Linux ne moremo na primer poganjati aplikacij sistema Windows.

Hramba podatkov

Razlika se pokaže pri hrambi podatkov. Tu rešitve za virtualizacijo uporabljajo precej podobne mehanizme. Kot navidezni disk znotraj navideznega stroja lahko pripnemo datoteke različnih formatov ali kar cel fizični nosilec. Ponudniki programske opreme za virtualizacijo imajo ponavadi poleg lastnega formata podporo vsaj še "surovim" oziroma "neobdelanim" (*raw*) datotekam ter nekaterim popularnejšim formatom (VMDK, qcow2), na voljo pa so tudi orodja, ki pretvorijo neki format v drugega. Razlika v formatih je v različnih funkcionalnostih, kot je npr. samodejno povečevanje navideznega diska, upoštevati pa je treba, da vsaka funkcionalnost doda k stroškom režije (razen pri neobdelanem formatu, ki takih funkcionalnosti nima). Pri tehnologiji vsebnikov ima tudi vsaka rešitev svojo izvedbo hrambe podatkov (ponavadi je za vsebnike namenjen določen direktorij na datotečnem sistemu gosta), vendar za zdaj podpore formatom različnih rešitev še ni.

Virtualizacija ima na tem področju prednost glede selitev iz ene tehnološke rešitve na drugo (npr. selitev iz VMWare na Xen). Pri taki selitvi lahko prestavimo datoteko z navideznim diskom iz navideznega stroja enega ponudnika v navidezni stroj drugega ponudnika in lahko že takoj nadaljujemo z delom (seveda privzemamo, da ne nastanejo težave z nastavitvami ali gonilniki). Pri tehnologiji vsebnikov (za zdaj) take možnosti nimamo.

5.2 Začasno ustavljanje, posnetki, kloniranje in migracija

Začasno ustavljanje

Začasna zaustavitev ali zamrznitev (*pause*, *freeze*) nam omogoča, da začasno ustavimo vse procese v gostu. Pri virtualizaciji imamo ponavadi možnost, da zamrznjeno stanje hranimo v pomnilniku ali shranimo na disk – tudi stanje pomnilnika navideznega stroja. To lahko primerjamo s spanjem (*sleep*) in hibernacijo (*hibernate*) fizičnega sistema.

Vsebniki imajo možnost ustavitve procesov, sama hramba stanja pomnilnika na disk pa še ni razširjena pri vseh rešitvah (ima jo OpenVZ, obstaja pa tudi projekt

CRIU [31], katerega cilj je vzpostavitev te funkcionalnosti za procese na splošno).

Posnetki in kloniranje

Posnetek (*snapshot*) je shranjeno stanje virtualnega diska ob določenem trenutku. Vse nadaljnje spremembe se potem hranijo v novi datoteki, tako da originalno stanje ostane nedotaknjeno. Posnetki omogočajo, da lahko podatke navideznega stroja/vsebnika prestavimo v različna stanja v času. Na voljo so tako pri rešitvah za virtualizacijo kot pri tehnologiji vsebnikov (npr. predloge, kot jih pozna Docker, so primer posnetkov), njihova uporaba pa je možna le z določenimi formati za navidezne diske (qcow2, vmdk...) ali z določenimi datotečnimi sistemi (LVM, aufs...).
Pri nekaterih rešitvah izraz posnetek označuje shrambo celotnega stanja gosta. Torej gre za kombinacijo posnetka diska (ali več njih) in shrambe stanja pomnilnika.

Kloniranje (*cloning*), tudi dupliciranje, je kopiranje obstoječega gosta. Ponavadi ima uporabnik izbiro "popolnega" klona, pri katerem se ustvari popolna kopija navideznega diska in je nastala kopija neodvisna od originala. V drugem primeru postopek kloniranja najprej ustvari posnetek obstoječega stanja gosta, novoustvarjena kopija pa za navidezni disk uporablja obstoječi posnetek, pri čemer se spremembe hranijo v novo datoteko. Ustvarjeni posnetek si lahko deli več klonov, pomembno je le, da zagotovimo dostop. Prednost takega "povezanega" klona (*linked clone*) je prihranek pri porabi prostora. Pogoji za uporabo povezanih klonov so podobni kot pri posnetkih, torej določen format datotek za navidezne diske oziroma določen datotečni sistem. Kloniranje je del funkcionalnosti obeh tehnologij.

Migracija

Migracija je postopek selitve gosta iz enega strežnika na drugega. Obstajata dve različici: "mrzla" (*cold migration*) in "vroča" ali "živa" migracija (*hot migration*, *live migration*). V prvem primeru je pri selitvi gost začasno ustavljen ali ugasnjen, sistem prenese stanje pomnilnika in podatkovnih medijev na nov strežnik in na novi lokaciji ponovno zažene gosta.
V drugem gre selitev "v živo", kar pomeni, da je med selitvijo gost aktiven. Postopek je v takem primeru bolj zapleten. Podatkovni mediji morajo biti vidni obema

strežnikoma, staremu in novemu, določene zaplete prinese s seboj tudi kopiranje vsebine pomnilnika gosta, ki se lahko med prenosom tudi spremeni, kar pomeni, da je treba te spremembe tudi upoštevati. Prekinitev delovanja gosta naj bi bila pri živi migraciji minimalna, tako da je uporabnik med delom sploh ne bi opazil.

Performančne zmogljivosti in stroški režije

Že eno izmed drugih poimenovanj za tehnologijo vsebnikov, lahkokategorna virtualizacija, nakazuje, da naj bi vsebniki manj obremenjevali gostiteljski sistem v primerjavi z navideznimi stroji, seveda pri enakih delovnih obremenitvah. Že če pogledamo stanje nekega navideznega stroja, v katerem razen operacijskega sistema ne teče noben drug proces. V takem primeru je še vedno aktivna emulacija strojne opreme, kar zahteva določeno število ciklov procesorja, da upravljanja pomnilnika navideznega stroja ne omenjamo. Ker pri vsebnikih tega ni, je gostiteljski sistem zagotovo manj obremenjen.
Primerjavo performančnih zmogljivosti in višino stroškov režije posameznih tehnologij na konkretnih področjih predstavljamo v poglavju 6.

Varnost in tveganja

Virtualizacija in tehnologija vsebnikov omogočata razdelitev fizične strojne opreme na več med seboj ločenih enot zaradi deljene rabe in možnega boljšega izkoristka. Pri tem je pomembno, da so te enote med seboj izolirane in ne vplivajo druga na drugo tudi zaradi varnosti. Kako je v primeru vdorov in napadov?
Pri virtualizaciji je glavna naloga ponudnika nekih storitev (npr. v oblaku), da poskrbi, da je gostiteljski sistem zavarovan pred vdori, napadi in zlonamerno kodo. Za varnost gosta (torej zaščita operacijskega sistema in aplikacij gosta) mora poskrbeti njegov uporabnik. Če pride do vdora v gosta, so gostitelj in preostali gosti še vedno izolirani in varni.
Kaj pa pri vsebnikih? Vsebniki tečejo kot procesi na gostiteljskem sistemu. Morebiten vdor v en vsebnik pomeni bistveno tveganje za gostiteljski sistem in preostale vsebnike. Kot smo že omenili, obstaja možnost, da napadalcu uspe izhod iz izolacije vsebnika in konča v operacijskem sistemu gosta. Zato je v takih primerih še posebej pomembna uporaba neprivilegiranih vsebnikov.

Poglavje 6

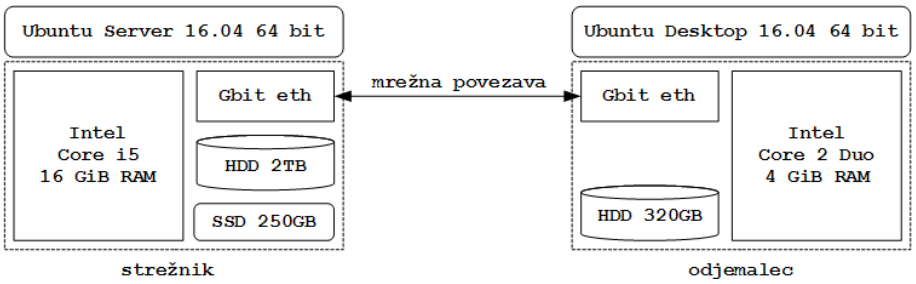
Testiranje

Za primerjavo virtualizacije in tehnologije vsebnikov smo na testnem računalniku namestili eno izmed izbranih rešitev za virtualizacijo ali tehnologijo vsebnikov (poglavje 6.2) in nato v navideznem stroju oziroma vsebniku izvajali teste (poglavje 6.3). Prvo testiranje je potekalo na strojni opremi, tako da smo dobili referenčne vrednosti testov. Za vsako izbrano rešitev smo namestili svoj operacijski sistem in v njem izbrano rešitev.

6.1 Poligon za testiranje

Naš testni poligon (slika 6.1) sestavljata dva računalnika.

1. Strežnik za virtualizacijo in tehnologijo vsebnikov (v nadaljevanju strežnik):
   - CPE: Intel Core i5 4460, 3.2–3.4 GHz, 4 jedra, 6 MiB predpomnilnika
   - pomnilnik: 16 GiB DDR3
   - trdi disk: 2 TB, 7200 obratov/min
   - SSD: 250 GB
   - mrežni vmesnik: 1 GBit
   - operacijski sistem: 64-bitni Ubuntu Server 16.04 (Linuxovo jedro 4.4.0)
2. Odjemalec za mrežne teste (v nadaljevanju odjemalec):
   - CPE: Intel Core 2 Duo E8200, 2.66 GHz, 2 jedri, 6 MiB predpomnilnika



Slika 6.1: Poligon za testiranje

- pomnilnik: 4 GiB DDR3
- trdi disk: 320 GB, 7200 obratov/min
- mrežni vmesnik: 1 GBit
- operacijski sistem: 64-bitni Ubuntu Desktop 16.04 (Linuxovo jedro 4.4.0)

Večina testov se izvaja le na strežniku, pri mrežnih testih pa potrebujemo tudi odjemalca.

6.2 Izbrane rešitve za virtualizacijo in tehnologijo vsebnikov

Hoteli smo primerjati vsaj dve virtualizacijski rešitvi in dve rešitvi za tehnologijo vsebnikov. Izbrali smo naslednje rešitve: KVM, Xen, LXC in Docker. Razlogi za izbiro so:

- Odprtokodne rešitve: vse štiri rešitve so na voljo brezplačno, brez omejitev za uporabo; pri odprtokodni skupnosti lahko najdemo številne nasvete in primere, kako konfigurirati in uporabljati posamezno rešitev.
- Uporaba v infrastrukturah za storitve v oblaku: izbrane rešitve so na voljo kot možna izbira hipervizorja v odprtokodni rešitvi za storitve v oblaku OpenStack, Amazon za svoj AWS uporablja sistem, zgrajen na podlagi Xena, Docker ponuja svojo rešitev Docker Cloud itn.

rešitev	**verzija**
Xen	4.6.0
KVM (QEMU-kvm)	2.5.0
Docker	1.10.3
LXC	2.0.0

Tabela 6.1: Uporabljene verzije rešitev

test	**verzija**
LINPACK	11.3.2
STREAM	5.10
RandomAccess (HPCC)	1.4.2
netperf	2.6.0
fio	2.2.10
bonnie++	1.97
UnixBench	5.1.3

Tabela 6.2: Verzije testov

- Preprosta namestitev (vse rešitve so del distribucije operacijskega sistema Ubuntu): posamezno rešitev smo lahko namestili preprosto z uporabo Ubuntujevih orodij za upravljanje paketov. Že sam Ubuntu je dobro dokumentiran, ima pa tudi dovolj veliko uporabniško skupnost, tako da bi na morebitne težave z delovanjem lahko hitro našli odgovor.

Verzije izbranih rešitev so navedene v tabeli 6.1. Pri vseh rešitvah smo izvajali teste (tabela 6.2) znotraj navideznega stroja/vsebnika, ki smo mu določili štiri navidezne CPE (oziroma pri vsebnikih nismo omejevali CPE) in velikost pomnilnika 8 GiB. Sledijo podrobnosti o izbranih rešitvah in nastavitvah navideznih strojev/vsebnikov.

6.2.1 KVM in Xen

Za obe rešitvi smo izbrali 64-bitno arhitekturo navideznega stroja, v katerega smo namestili operacijski sistem Ubuntu Server 16.04.

Pri hipervizorju Xen smo se odločili za dva testa, in sicer smo en navidezni stroj konfigurirali kot HVM (Xenova oznaka za strojno podporo virtualizaciji), drugega pa kot PVH (kombinacija paravirtualizacije s strojno podporo virtualizaciji). Prvi nam je tako služil tudi za primerjavo s KVM, drugi naj bi vsaj na papirju združeval prednosti virtualizacije in paravirtualizacije (upoštevati moramo, da je razvoj PVH v fazi ponovne izgradnje arhitekture, tako da je pričakovati spremembe pri tem načinu delovanja).
Konfiguracijske datoteke so v prilogah A.1, A.2 inA.3. Pri obeh rešitvah smo uporabili paravirtualizirane gonilnike za V/I naprave (*virtio* pri KVM, *PV drivers* pri Xen). Poleg navideznega diska – datoteke vrste *raw* smo navideznim strojem pripeli še SSD, tudi v "surovi" obliki, tako da je imel vsak navidezni stroj neposreden dostop do njega.

6.2.2 LXC in Docker

Pri obeh vsebnikih smo uporabili predlogo Ubuntu Xenial (torej verzijo 16.04), 64-bitno varianto (Docker podpira samo 64-bitne gostiteljske sisteme). SSD smo pripeli v gostiteljski sistem v izbrani direktorij, ki smo ga potem pripeli v vsebnik. LXC pri zagonu vsebnika prebere posebne konfiguracijske datoteke (priloga A.4), Docker pa vse nastavitve sprejme ob zagonu (priloga A.5). Pri Dockerjevem vsebniku smo morali izpostaviti še zaporedje vrat za mrežne teste, mrežne nastavitve pa smo nastavili s posebno skripto (tudi v prilogi).

6.3 Izvedba testov in rezultati

Kot že omenjeno v predhodnih poglavjih prinašata virtualizacija in tehnologija vsebnikov s seboj določene stroške režije, povezane z delovanjem in upravljanjem navideznih strojev oziroma vsebnikov. Zanimalo nas je, kolikšni so ti stroški v primerjavi s fizičnim sistemom in za koliko manjši (če sploh) naj bi bili ti stroški pri tehnologiji vsebnikov v primerjavi z virtualizacijo. Iz rezultatov smo želeli dobiti informacije, na podlagi katerih bi se lažje odločili za uporabo določene tehnologije na določenem področju. Izbrali smo teste, ki obravnavajo različne računalniške vire: računske (zmogljivost CPE), pomnilniške (dostop do pomnilnika), omrežne (odzivnost in pasovna širina) in podatkovne (branje in pisanje). Ustrezne vnosne

datoteke oziroma zagonski argumenti za teste so v prilogi A.6.

6.3.1 Računski test – Intel Optimized LINPACK Benchmark

Intel Optimized LINPACK Benchmark [39] je Intelova verzija testa LINPACK 1000, ki reši sistem linearnih enačb ($Ax = b$), zmeri čas, ki ga potrebuje za rešitev sistema, in pretvori ta čas v vrednost, ki označuje performančno zmogljivost sistema. V tem primeru so to milijarde operacij s plavajočo vejico na sekundo, *Gflops*. Ta test smo torej izbrali, ker nas zanima računska moč sistema.

Intelova verzija testa uporablja Intel Math Kernel Library [38], optimizirano matematično knjižnico, katere uporaba naj bi zagotovila hitrejše matematično procesiranje na Intelovi in drugih kompatibilnih arhitekturah.
Test sprejema več vnosnih parametrov, med drugim lahko definiramo, da požene več različnih testiranj z različnimi velikostmi problema in različnim številom iteracij za vsako testiranje. Pri tem moramo paziti na pomnilnik, ki je na voljo, saj poraba zelo hitro raste, ko povečujemo velikost problema.
Za test smo izbrali naslednje parametre:

- število enačb: 31000
- velikost matrike: 31000
- število ponovitev: 10

Poraba pomnilnika pri takih parametrih je potem približno $31000 * 31000 * 8$ B oz. 7,16 GiB, kar ne presega 8 GiB, namenjenih navideznemu stroju oz. vsebniku in pusti na voljo dovolj pomnilnika za operacijski sistem.

Omeniti moramo težavo, ki smo jo imeli s tem testom. Intelov test pred izvajanjem meritev najprej pridobi informacije o strojni opremi, to je, koliko procesorjev, jeder in niti ima sistem, in se temu ustrezno prilagodi. Test je pravilno zaznal štiri jedra in štiri niti v vseh vsebnikih in navideznih strojih, razen v navideznem stroju znotraj Xen PVH, kjer je zaznal le eno jedro in eno nit. Tu velja omeniti, da je sam operacijski sistem znotraj navideznega stroja pravilno zaznal štiri (virtualne)

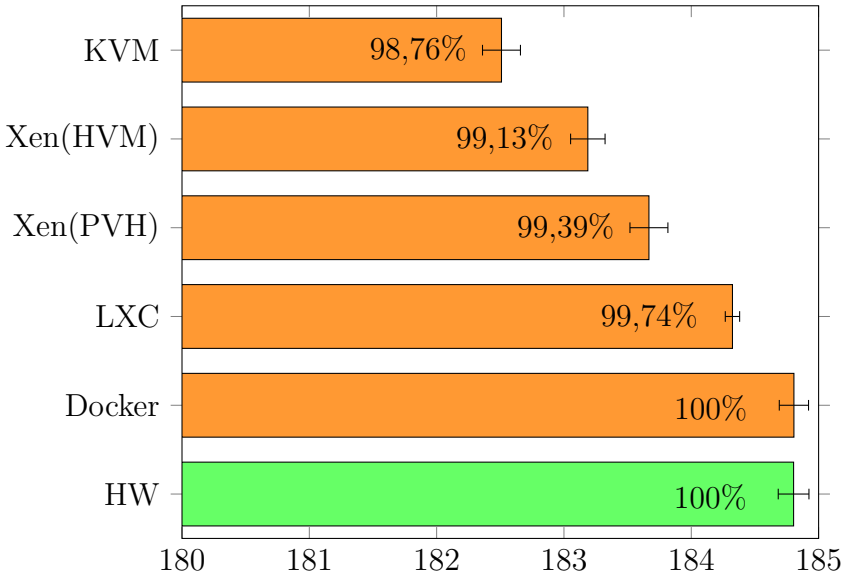
CPE. Po poizvedovanju na dopisnem seznamu uporabnikov Xena, smo dobili odgovor in rešitev. Xen PVH ima drugačno implementacijo predstavitve procesorja in topologije v primerjavi s Xen HVM (kjer ni bilo težav), zato je bilo treba dodati še eno nastavitev v konfiguracijo navideznega stroja. Ta nastavitev je dodala informacijo o številu jeder in test je potem deloval po pričakovanju.

Rezultati

Pri računskem testu smo pričakovali majhne razlike med navideznimi stroji in vsebniki oziroma majhna odstopanja glede na fizični sistem. Pri takem testu ni nekih sistemskih klicev, kot so preklapljanja med procesi. Gre za neposredno izvrševanje matematičnih ukazov, zato nismo pričakovali velikih stroškov režije.
Rezultati so naša pričakovanja potrdili. Slika 6.2 nam prikazuje primerjavo med sistemi z vrednostmi v milijardah operacij s plavajočo vejico na sekundo (Gflops). Kako blizu so si rezultati, kažejo tudi relativne vrednosti v grafu glede na fizični sistem (označen s HW, obarvano zeleno). Zmagovalec testa je Docker, katerega povprečna vrednost Gflops je enaka fizičnemu sistemu, a tudi preostali sistemi zaostajajo izredno malo (najslabši med njimi, KVM, manj kot 1,5 odstotne točke). Če upoštevamo performančne zmogljivosti izvajanja le računsko zahtevnih (matematičnih) operacij, ne moremo glede na rezultate v ospredje postaviti le ene rešitve ali tehnologije. V takem primeru o izbiri tehnologije odločajo drugi dejavniki.
Popolni rezultati meritev so v prilogi, tabeli B.1 in B.2.

6.3.2 Test pasovne širine pomnilnika – STREAM

STREAM [47] je program za merjenje pasovne širine pomnilnika (*memory bandwidth*). Njegov avtor navaja, da je STREAM *de facto* standard za te vrste meritev. Na voljo je njegova izvorna koda, je preprost za uporabo, podpira eno- in večprocesorske sisteme in je sorazmerno razširjen (o njegovi uporabi pišejo v člankih o primerjavah performančnih zmogljivosti [10, 28, 19], je pa tudi del nabora HPCC [35]).
STREAM je na voljo v obliki izvorne kode v jezikih C in Fortran, ki jo prevedemo sami, pri čemer lahko podamo vrednosti za določene predprocesorske spremenljivke, kot je npr. velikost polja, s katerim dela STREAM. STREAM podpira



Slika 6.2: Intel MKL LINPACK - rezultati (Gflops)

eno- in večprocesorske sisteme ter tudi paralelne sisteme. STREAM v eni iteraciji opravi štiri meritve, in sicer:

- kopiranje (*copy*): tu gre za preprosto operacijo branja iz enega polja in pisanje v drugo polje ($a[i] = b[i]$)
- skaliranje (*scale*): podobno kot pri kopiranju, le da je vmes še operacija množenja ($a[i] = q * b[i]$)
- vsota (*add*): tu gre za dve branji in eno pisanje ($a[i] = b[i] + c[i]$)
- triada (*triad*): podobno kot pri vsoti, le da je vmes še operacija množenja ($a[i] = b[i] + q * c[i]$)

STREAM izvede več iteracij vseh meritev in vrne najboljši rezultat za vsako meritev.

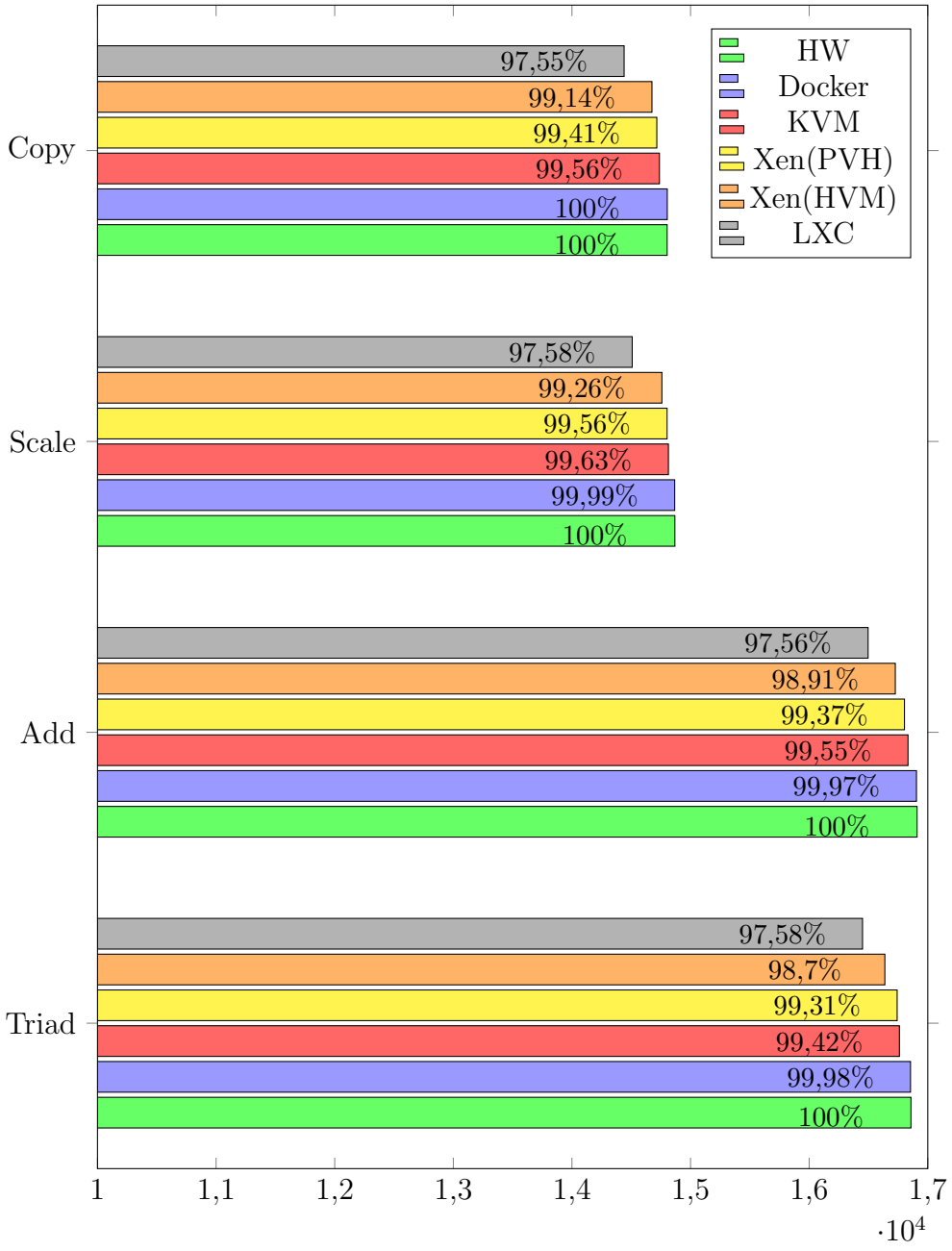
V našem primeru smo vključili generiranje vzporedno izvršujoče kode in nastavili velikost polja na 100 000 000, pri kateri je poraba pomnilnika približno 2,2 GiB. Tu nismo ciljali na porabo 8 GiB, saj gre pri testu STREAM za 32-bitno kodo, tako da lahko pri preveliki izbiri velikosti polja pride do preliva (*overflow*). Prav tako ta velikost močno presega priporočeno velikost polja, ki znaša štirikratno velikost procesorjevega medpomnilnika L3.

Rezultati

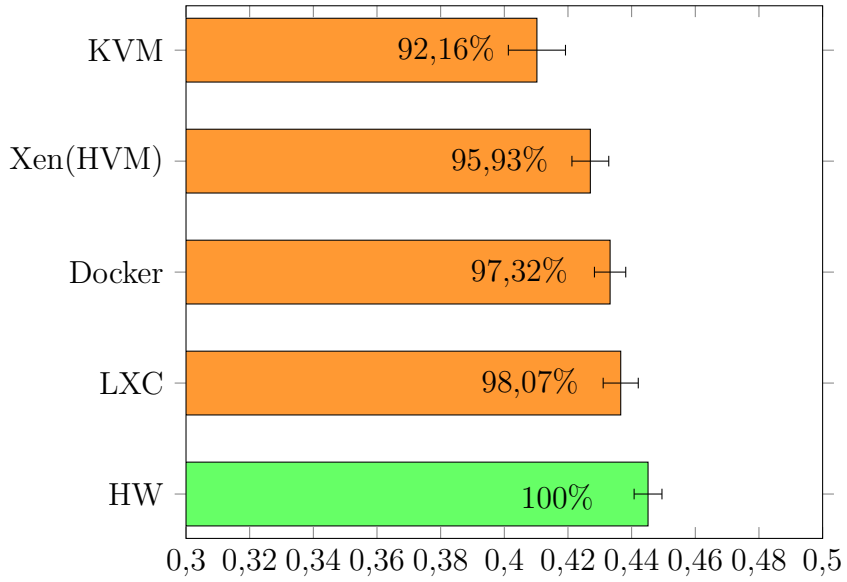
Zaradi načina delovanja testa STREAM ni veliko zgrešitev v TLB, ki pri navideznih strojih stanejo veliko – posledično nismo pričakovali velikega poslabšanja rezultatov navideznih strojev v primerjavi s fizičnim sistemom.
Pri vseh štirih meritvah si rezultati sistemov sledijo v istem vrstnem redu (slika 6.3), s podobnimi odstopanji. Podobno kot pri testu LINPACK je tudi v tem primeru absolutni zmagovalec Docker, presenetilo pa nas je zadnje mesto LXC. Tu gre morda za razlike v implementaciji omejitve pomnilnika na nastavljenih 8 GiB v primerjavi z Dockerjem.
Popolni rezultati meritev so v prilogi, tabeli B.3 in B.4.

6.3.3 Test naključnega dostopa do pomnilnika – RandomAccess

Tako kot STREAM je tudi RandomAccess eden izmed testov v naboru testov HPC Challenge (HPCC [35]), ki merijo performančno zmogljivost sistema, ki je lahko bodisi en sam računalnik bodisi več med seboj povezanih računalnikov. Tu gre za sedem testov, ki jih žal brez posega v izvorno kodo ni moč izvrševati posamično, zato smo izvajali iteracije celotnega testa (žal za razliko od testa STREAM RandomAccess ni na voljo samostojno). RandomAccess meri naključni dostop do pomnilnika, pri čemer rezultat dobimo v milijardah ažuriranj na sekundo (Giga UPdates per Second – GUPS).
Podobno kot pri testu LINPACK, smo imeli tudi tukaj težavo pri testiranju navideznega stroja v testu Xen PVH. Kot kaže tudi HPCC pred meritvami preverja topologijo sistema, preden požene testiranje. V primeru Xen PVH je to testiranje povzročilo sesutje testa (napaka je bila poskušanje deljenja z nič). Napako smo sporočili na dopisni seznam uporabnikov Xena, Xen PVH pa izpustili iz tega dela testiranja.
Omeniti moramo tudi odstopanje, ki se je pojavilo pri eni izmed meritev v navideznem stroju v testu Xen HVM (tabela B.5, označeno z rdečo). Ker je šlo za več kot 60-odstotno odstopanje od povprečja, smo to meritev izločili. Menimo namreč, da je v tem primeru najverjetneje prišlo do obremenitve sistema, ki ni povezana z regularnim delovanjem hipervizorja.



Slika 6.3: STREAM - rezultati (MB/s)



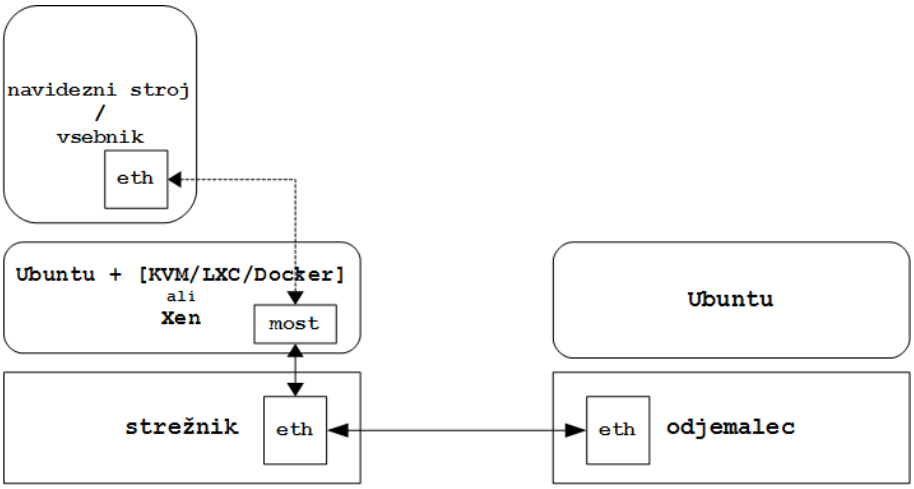
Slika 6.4: RandomAccess - rezultati (GUPS)

Rezultati

RandomAccess je zasnovan tako, da povzroča zgrešitve v TLB, zato imata pričakovano najslabše rezultate (slika 6.4) obe virtualizacijski rešitvi, KVM in Xen. KVM ima poleg najslabšega povprečja tudi najbolj razpršene meritve, po drugi strani pa Xen niti ne zaostaja veliko za obema vsebnikoma. Situacija z LXC je tudi obrnjena v primerjavi s testom STREAM, saj predhodni poraženec pri testu RandomAccess vodi.
Popolni rezultati meritev so v prilogi, tabeli B.5 in B.6.

6.3.4 Test odzivnosti omrežnega vmesnika – netperf

Za mrežne teste smo potrebovali še odjemalca, ki je pošiljal testne podatke navideznemu stroju/vsebniku na strežniku preko lokalnega omrežja. Odjemalec in strežnik sta bila neposredno povezana med seboj. Navidezne stroje in vsebnike smo umestili v isto omrežje preko mostu (*Linux bridge*, uporabili smo orodje *bridge-utils*). Most je povezoval fizični omrežni vmesnik strežnika z navideznim omrežnim vmesnikom navideznega stroja/vsebnika (slika 6.5).

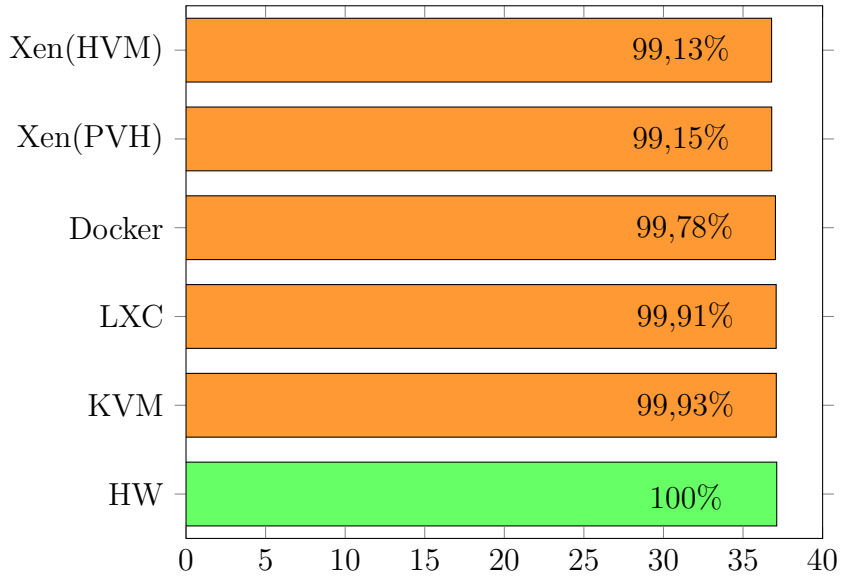


Slika 6.5: Postavitev mrežnega testa

Za test odzivnosti smo uporabili merilni program netperf [49]. Netperf ima na izbiro več različnih meritev, mi smo izbrali dve. Prva meritev je vrste zahteva/odgovor (*request/response, RR*), pri čemer netperf meri število transakcij na sekundo. Transakcijo sestavljata ena zahteva in en odgovor, iz rezultata potem lahko izračunamo tudi povprečno zakasnitev (*latency*). Druga meritev je vrste povezava/zahteva/odgovor (*connect/request/response, CRR*), ki je dejansko enaka prvi, le da netperf pri vsaki transakciji odpre novo povezavo in jo na koncu zapre. V primerjavi s prvo meritvijo nam je druga pokazala, kolikšne stroške režije v povprečju prinaša odpiranje povezave.

Rezultati

Rezultati meritev testa zahteva/odgovor (slika 6.6) kažejo, da so sistemi precej enakovredni. Zmagovalec testa je KVM, vendar so preostali sistemi manj kot odstotno točko zadaj. Malce drugačno sliko kažejo rezultati testa povezava/zahteva/odgovor (slika 6.7). Prva tri mesta so premešana, vendar manj kot 0,2 odstotne točke narazen. Malce je presenetil rezultat obeh Xen-ovih navideznih strojev, ki zaostajata za dobre štiri odstotne točke glede na fizični sistem. Kot kaže, ima odpiranje in zapiranje povezave večji vpliv na performančne zmogljivosti mrežnega vmesnika v primerjavi s KVM-jevim navideznim strojem, ki tako kot oba Xenova stroja uporablja paravirtualiziran mrežni vmesnik.



Slika 6.6: Netperf – zahteva/odgovor – rezultati (št. zahtev/s)

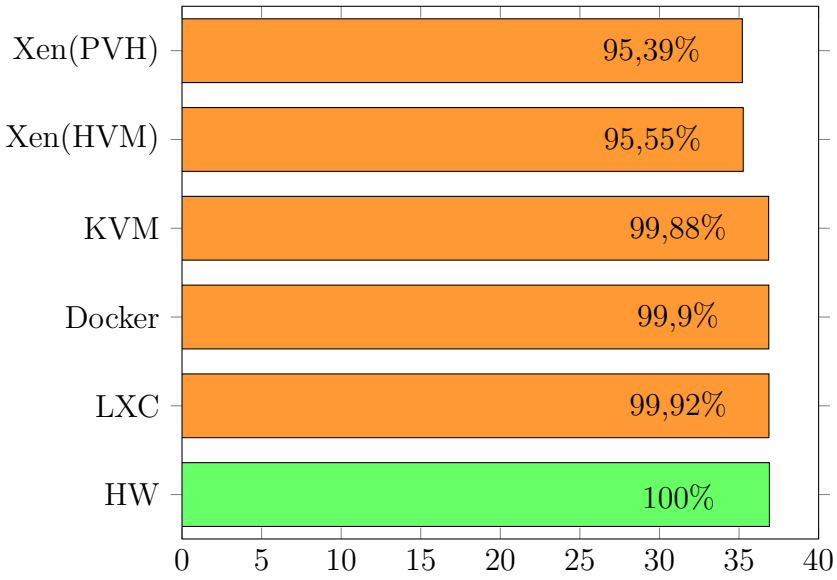
Popolni rezultati meritev so v prilogi, tabele B.7, B.8, B.9 in B.10.

6.3.5 Test pasovne širine omrežnega vmesnika – fio

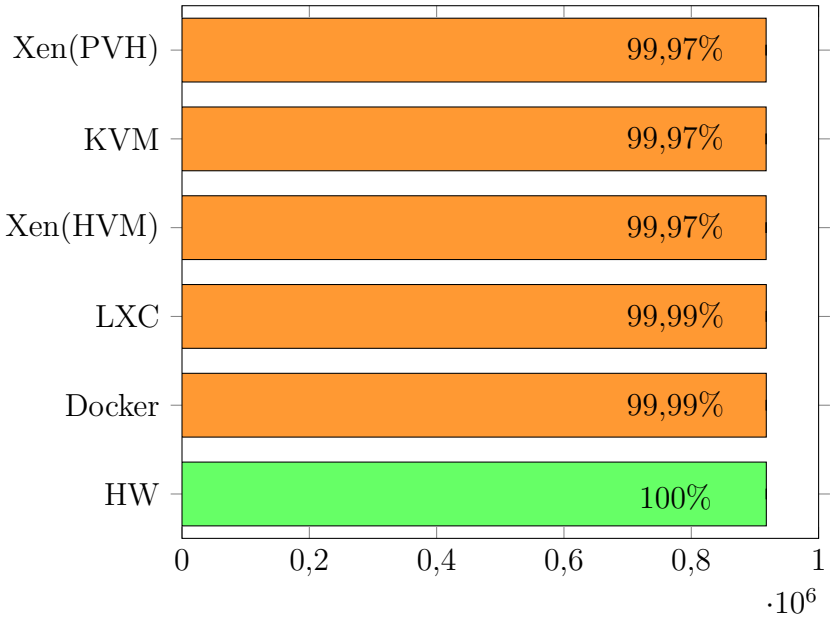
Program fio (*flexible I/O tester*) opravlja različne naloge (lahko tudi več hkrati) na V/I napravah in vrne rezultate izvajanja. V našem primeru smo fio uporabili za merjenje pasovne širine mrežnega vmesnika. Programu fio smo podali, da prenese od odjemalca k strežniku datoteko velikosti 30 GiB s hitrostjo prenosa 200 MiB/s, kar je več od teoretično maksimalne možne hitrosti prenosa preko gigabitne povezave.

Rezultati

Rezultati na sliki 6.8 kažejo zanemarljive razlike med sistemi. Vendar so se spet pojavile posebnosti pri Xenovih navideznih strojih. Tabeli B.11 in B.12 v prilogi vsebujeta rezultate vseh meritev. Fio pri posameznem rezultatu vrne povprečno pasovno širino in tudi največjo pasovno širino izvajanja. Pri vseh preostalih sistemih je bila največja pasovna širina enaka za vse iteracije, pri obeh Xenovih navideznih strojih pa se je spreminjala in nekajkrat tudi močno presegla tudi teoretično



Slika 6.7: Netperf – povezava/zahteva/odgovor – rezultati (št. zahtev/s)



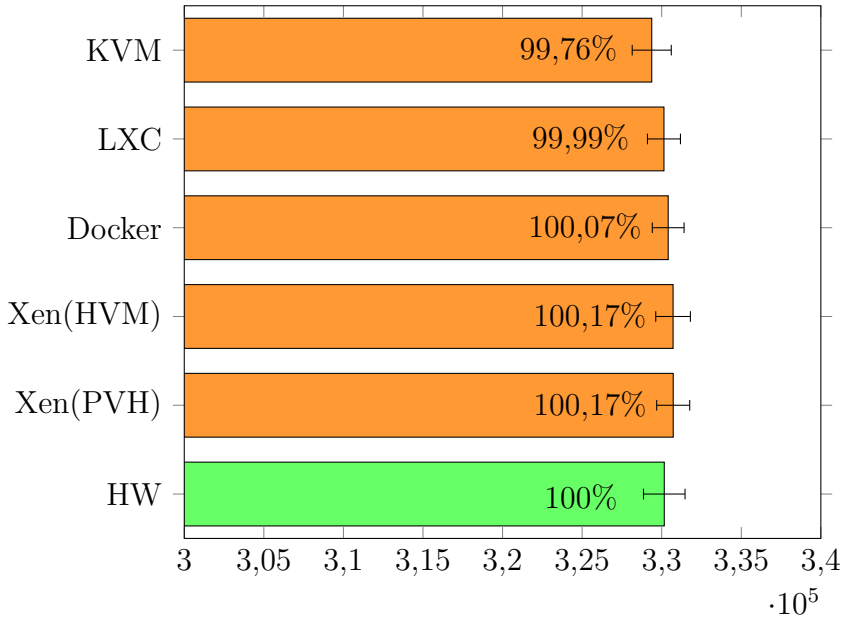
Slika 6.8: fio – rezultati (Kbit/s)

možno pasovno širino. V teh primerih je odstopalo potem tudi povprečje. Po našem mnenju je to odstopanje navzgor posledica delovanja paravirtualiziranega vmesnika, ki verjetno zaradi uporabe medpomnilnika kaže previsoke vrednosti. Meritve, ki so odstopale smo izločili iz izračunov povprečja.

6.3.6 Test V/I naprav – bonnie++

Bonnie++ [30] je program za merjenje performančnih zmogljivosti trdih diskov (in drugih medijev za shranjevanje podatkov) in datotečnega sistema. Test opravi več meritev: sekvenčno pisanje in branje velike datoteke, sekvenčno in naključno ustvarjanje, preverjanje in brisanje veliko majhnih datotek, meri tudi obremenitve procesorja in zakasnitve (*latency*). Bonnie++ sprejema več parametrov ob zagonu, med drugim tudi velikost datoteke za pisanje in branje ter število datotek (večkratnik 1024) za preostale meritve. Po priporočilu samega testa, da mora biti datoteka za sekvenčni test vsaj dvakrat večja od velikosti sistemskega pomnilnika, smo določili velikost 32 GiB, za ostale teste pa smo določili generiranje $128 * 1024$ datotek velikosti 1 KiB (pri manjših vrednostih bonnie++ na našem sistemu ni mogel pridobiti dovolj natančnih rezultatov in jih potem ne prikaže).

Za testiranje smo uporabili disk SSD velikosti 250 GB, s čimer smo se izognili vplivom vrtenja plošč na performančne zmogljivosti, do katerih prihaja pri klasičnih trdih diskih. Na disku smo ustvarili particijo vrste *ext4* (ki je zajemala celoten prostor), do katere je vsak sistem dostopal na svoj način. Navidezni stroji v KVM in Xen so dostopali neposredno do diska preko paravirtualiziranih gonilnikov in priklopili (*mount*) particijo znotraj gostujočega operacijskega sistema. LXC in Docker imata vsak svojo različico dostopa do neke zunanje particije, vendar na podoben način. V obeh primerih smo particijo priklopili na izbran direktorij gostiteljskega sistema. Pri LXC smo nato v nastavitvah podali, v kateri direktorij znotraj vsebnika naj priklopi izbrani direktorij, pri Dockerju pa smo izbrani direktorij podali kot podatkovni nosilec (*data volume*). Velja omeniti, da se z uporabo podatkovnega nosilca izognemo stroškom režije, ki jih sicer ima Docker pri podatkovnih dostopih zaradi svojega večplastnega datotečnega sistema.

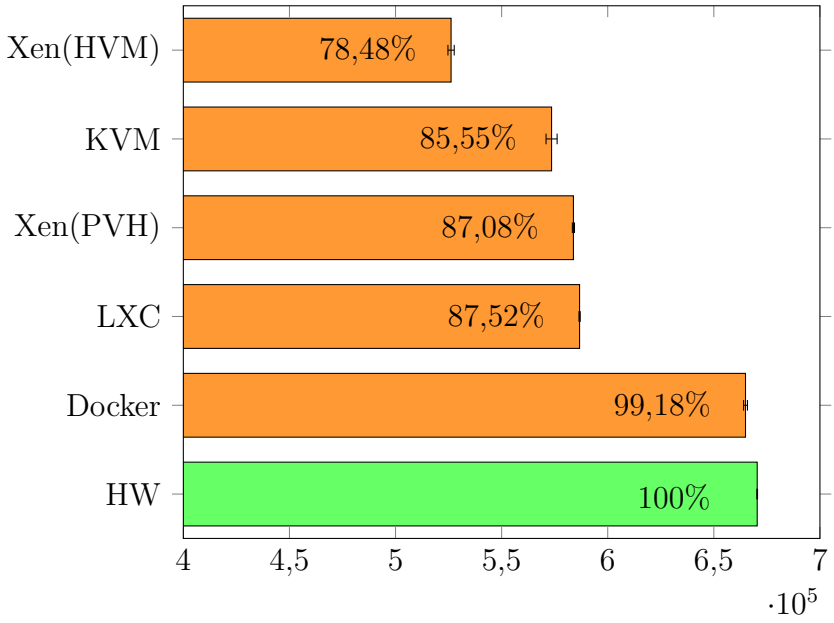


Slika 6.9: bonnie++ - rezultati,
sekvenčno pisanje (KB/s)

Rezultati

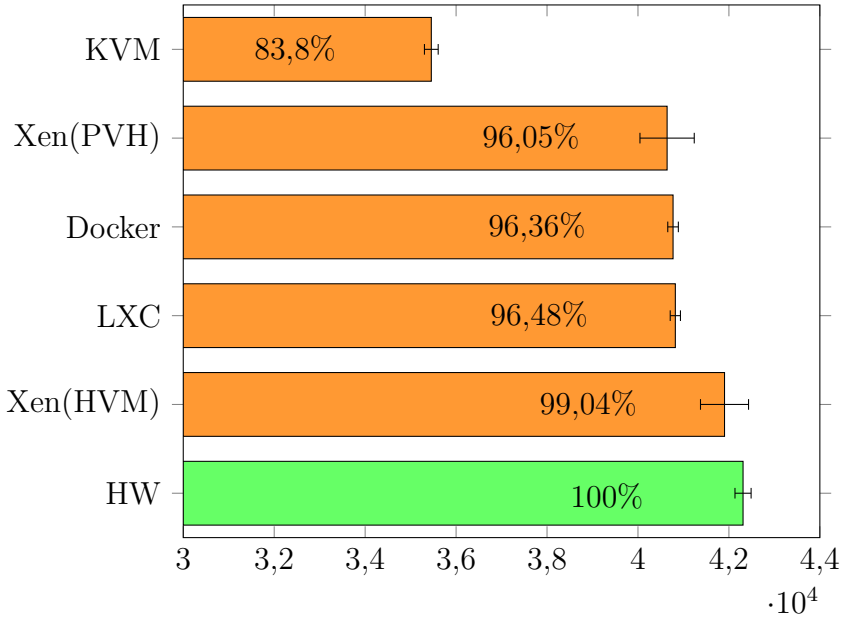
Od vseh meritev smo zbrali rezultate sekvenčnega pisanja in branja velike datoteke ter rezultate naključnega kreiranja in preverjanja majhnih datotek. Najbolj konsistentno glede vseh sistemov je bilo sekvenčno pisanje velike datoteke (slika 6.9), pri čemer so rezultati pisanja vseh sistemov pravzaprav enaki rezultatu pisanja fizičnega sistema. Rezultati sekvenčnega branja velike datoteke so že pokazali razlike v sistemih (slika 6.10), zaostajali so vsi sistemi razen Docker-ja. Poleg stroškov režije pri navideznih strojih gre verjetno tudi za različno uporabo medpomnjenja pri branju. Presenetil nas je rezultat LXC-ja, ki uporablja podoben način dostopa do particij kot Docker, kot kaže pa z večjimi stroški režije ali tudi z drugačno uporabo medpomnenja pri branju.

Naključno ustvarjanje datotek kaže drugačno sliko (slika 6.11), na vrhu je Xenov navidezni stroj različica HVM. Sledijo mu vsebnika in Xenov navidezni stroj različica PVH, na zadnjem mestu pa je KVM s precejšnjim zaostankom. Naključno preverjanje datotek uporablja Linux-ov ukaz *stat* [42], ki lahko prikaže različne informacije o datoteki (ime, velikost, lastništvo, dostopne pravice idr.).

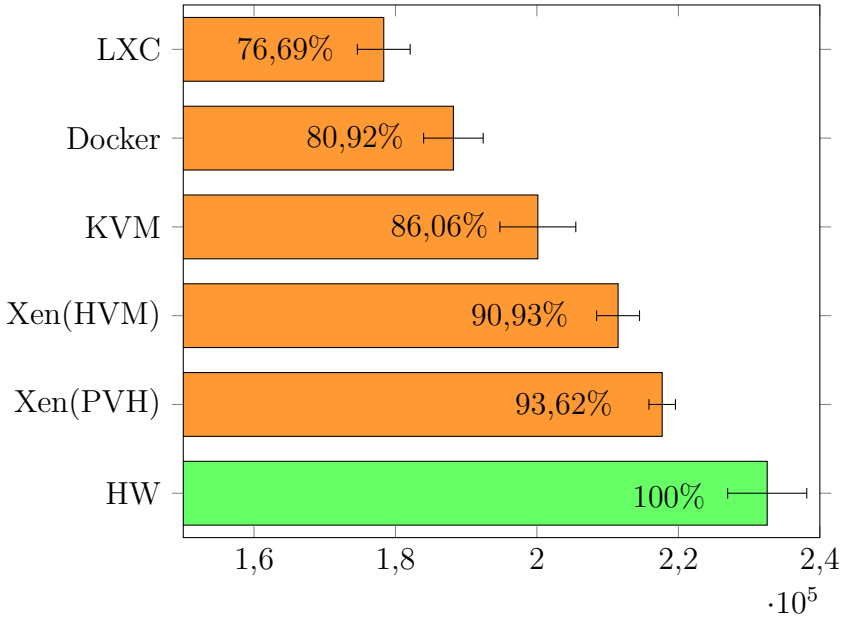


Slika 6.10: bonnie++ - rezultati,
sekvenčno branje (KB/s)

Presenetljivo sta bila pri tem testu vsebnika na zadnjem mestu (slika 6.12), glede na to, da je *stat* sistemski klic, bi pričakovali najslabše rezultate pri navideznih strojih. Popolni rezultati meritev so v prilogi, tabeli B.13 in B.14.



Slika 6.11: bonnie++ - rezultati,
naključno kreiranje (št. datotek/s)



Slika 6.12: bonnie++ - rezultati,
naključno preverjanje (št. datotek/s)

6.3.7 UnixBench

UnixBench [55] sestavlja množica testov, s katerimi pridobimo oceno performančnih zmogljivosti Unixu podobnega sistema. Rezultate testov Unixbench primerja z rezultati osnovnega sistema (SPARCstation 20-61 iz leta 1983) in na koncu poda indeks zmogljivosti za vsak test ter skupni indeks za celoten sistem. Pri sistemih z več CPE izvede Unixbench najprej eno instanco testov, nato pa še N instanc testov naenkrat (N = število CPE). Čeprav je Unixbench nastal pred več kot 30 leti, so testi še vedno aktualni. Skozi leta so ga različni razvijalci vzdrževali in izboljševali, prosto dostopna koda pa da tudi natančen vpogled, kako Unixbench-evi testi delujejo, s tem pa se izniči tudi možnost manipulacije z rezultati.
Tu velja omeniti, da avtorji opozarjajo, da Unixbench meri lastnosti celotnega sistema, tako rezultati niso odvisni le od strojne opreme, ampak tudi operacijskega sistema, knjižnic in prevajalnika.

Testi, ki jih izvaja UnixBench, so naslednji:

- Whetstone in Drhystone: Gre za sintetična testa, namenjena meritvam in primerjavi zmogljivosti računalnikov. Zasnovana sta tako, da obremenita CPE na podoben način, kot neka tipična množica programov (npr. znanstvene aplikacije). Whetstone meri hitrost in učinkovitost operacij s plavajočo vejico, Drhystone pa izvaja le celoštevilske operacije.

- Izvajanje klica *execl*: Test meri število klicev execl, ki je eden izmed ukazov iz družine exec. Execl ustvari nov proces znotraj konteksta obstoječega procesa, pri čemer novi proces zamenja obstoječega.

- Kopiranje datotek: Test meri, kako hitro lahko sistem kopira podatke iz ene datoteke v drugo, pri čemer uporablja različne velikosti medpomnilnika in blokov (medpomnilnik/blok: 1024B/2000B, 256B/500B, 4096B/8000B).

- Komunikacija z uporabo cevi (*pipe*): Cev je najpreprostejša oblika komunikacije med procesi. Test meri, kako hitro proces zapiše 512 B v cev in jih prebere nazaj.

- Izmenjava podatkov med dvema procesoma: Test meri, kolikokrat si lahko dva procesa izmenjata podatke skozi cev. Za razliko od predhodnega testa,

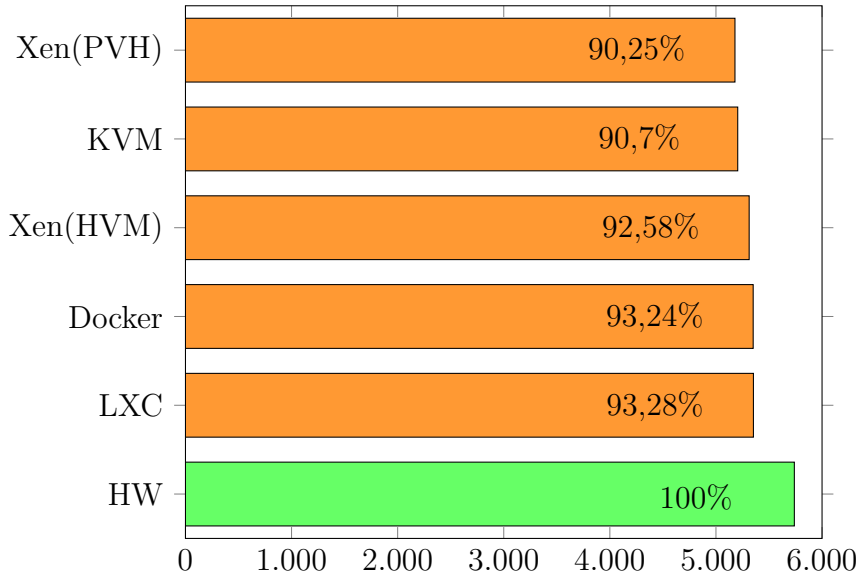
je ta podoben obnašanju realne aplikacije.

- Kreiranje procesa: Test meri kolikokrat nek proces z uporabo ukaza *fork* ustvari in uniči ustvarjeni proces.
- Izvajanje skript: Test meri, kolikokrat na minuto nek proces požene in ustavi določeno število skript, najprej po eno, nato še osem sočasno.
- Stroški režije izvajanja sistemskih klicev: Test oceni stroške režije izvajanja sistemskega klica. V tem primeru gre za preprost program, ki izvaja klic *getpid*, ki vrne oznako kličočega procesa.
- Grafični testi: UnixBench vsebuje tudi nekaj grafičnih testov, ki jih nismo izvajali.

Unixbench smo izvajali na disku SSD, pripetem v navidezne stroje/vsebnike na enak način kot pri izvajanju testa bonnie++ in iz enakih razlogov (izogibanje stroškom režije zaradi vrtenja plošč in uporabe večplastnega datotečnega sistema pri Docker-ju).

Rezultati

Vsi rezultati so vrste "večje je boljše".
Skupni indeks (slika 6.13) kaže vodstvo vsebnikov, vendar je Xen HVM zanemarljivo zadaj. Bolj nas je zanimalo, kako so se sistemi odrezali pri posameznih testih. Niso nas zanimali rezultati računskih testov in testov kopiranja datotek, saj smo ta področja obdelali že z drugimi testiranji, osredotočili smo se na teste, pri katerih so v uporabi sistemski klici: izvajanje klica *execl*, komunikacija med procesoma, kreiranje procesa in stroški režije pri sistemskih klicih (na sliki 6.14 označeni z Execl, Kom., Kreir., Rež., po vrsti).
Že takoj je izstopal rezultat testa stroškov režije, saj vrednosti pri obeh Xenovih navideznih strojih presegata vrednost pri fizičnem računalniku. Odstopanje je dovolj veliko, da ga ne moremo imeti za statistično napako. Brez podrobne analize obnašanja hipervizorja Xen, je težko oceniti razloge za boljši rezultat. Test je, kot že prej omenjeno, sorazmerno preprost, gre za ponovljeno izvajanje klica *getpid*, za katerega dokumentacija [41] omenja, da sistemska knjižnica (*glibc*) pri zaporednih klicih v medpomnilniku hrani oznako procesa, da se izogne več zaporednim

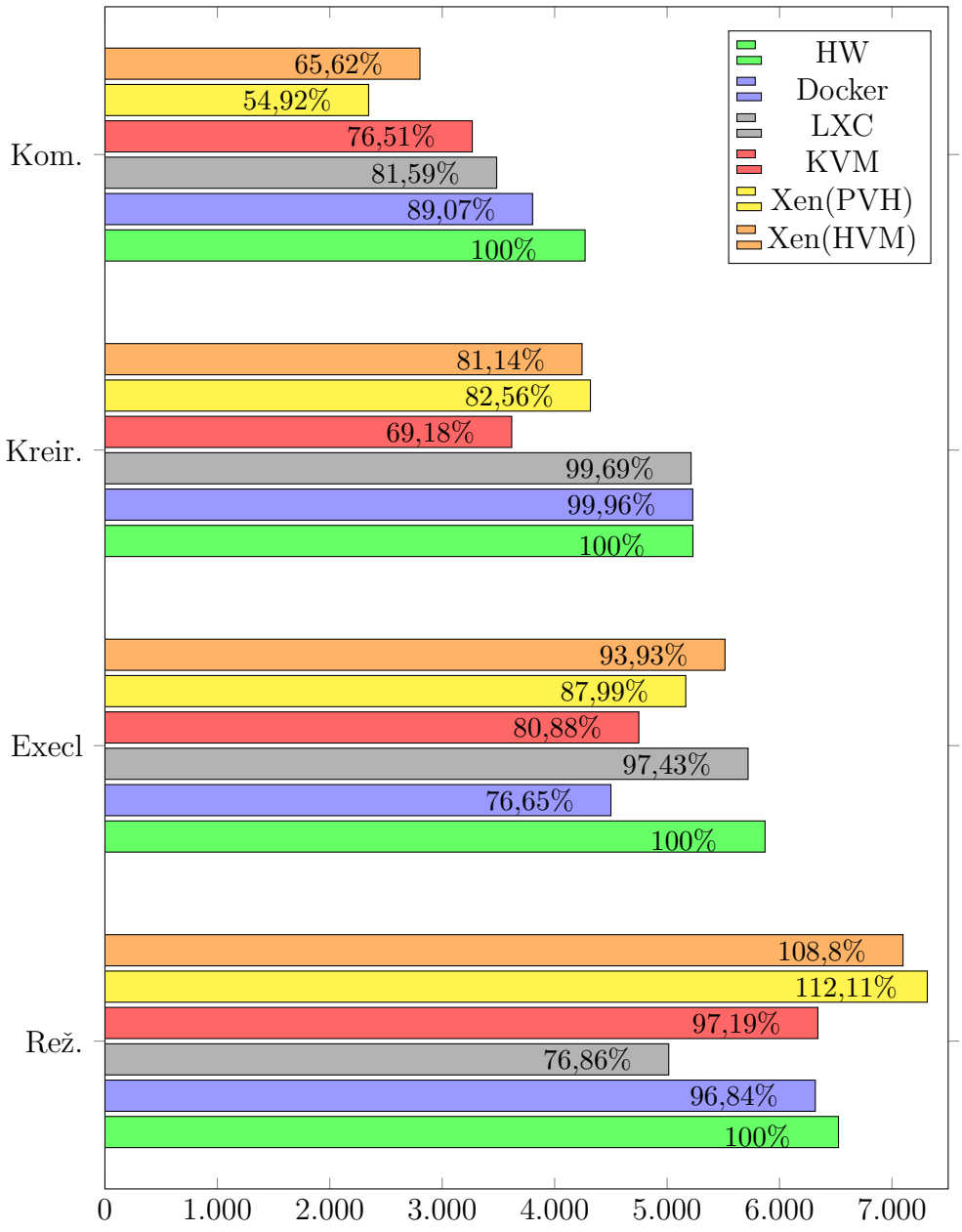


Slika 6.13: UnixBench - skupni indeks

sistemskim klicem. Na podlagi rezultatov in omenjene uporabe medpomnilnika je naše mnenje, da ta test ne kaže prave slike ocene stroškov režije sistemskih klicev. Pri preostalih testih sta najboljše rezultate pokazala vsebnika (izjema je klic *execl* pri Dockerju, ki ima najslabši rezultat), navidezni stroji pa si menjavajo mesta pri posameznih testih.

Slabše rezultate pri klicih s sistemskimi testi pri navideznih strojih smo pričakovali, kot kažejo rezultati, imajo z večjimi stroški režije pri določenih klicih težave tudi vsebniki.

Popolni rezultati meritev so v prilogi, tabele B.15, B.16, B.17, B.18, B.19 in B.20.



Slika 6.14: Unixbench - rezultati testov s sistemskimi klici

6.4 Diskusija

Ob izvedbi primerjalnih testov med navideznimi stroji in vsebniki smo predvidevali, da bo tehnologija vsebnikov pokazala boljše rezultate od virtualizacije. Manjši stroški režije tehnologije vsebnikov pomenijo manjšo obremenitev gostiteljskega sistema, kar vsaj teoretično pomeni, da je na voljo več virov za vsebnike.
Naša predvidevanja so se delno tudi potrdila. Računski in pomnilniški testi so res pokazali boljše rezultate pri vsebnikih, a pri tem ne moremo govoriti o neki veliki prednosti – še več, v določenih primerih lahko prednost vsebnikov pred navideznimi stroji označimo za zanemarljivo, pri testu STREAM pa je LXC celo najslabši.
Omrežni testi so nam pokazali, da sta tehnologiji izenačeni (malce slabše rezultate ima le Xen pri testu vrste povezava/zahteva/odgovor), na podlagi tega lahko potegnemo sklep, da je to posledica uporabe paravirtualiziranih gonilnikov za omrežne vmesnike namesto emulacije.
Pri testih branja in pisanja na disk ne moremo govoriti niti o izenačenju niti o zmagovalcu, saj so bila na vsakem področju mesta premešana. Predvidevamo, da so te razlike povezane z uporabo medpomnilnika (*caching*) in drugih nastavitev, tako da je po našem mnenju še prostor za optimizacijo.
Tudi pri rezultatih UnixBencha ne moremo tehnologije vsebnikov razglasiti za absolutnega zmagovalca. Rezultati so nam pričakovano pokazali šibko točko virtualizacije – sistemske klice, a kaže pa imajo težave z določenimi klici tudi vsebniki (test *execl* pri Dockerju).

Če povzamemo rezultate, je tehnologija vsebnikov glede performančnih zmogljivosti v rahli prednosti pred virtualizacijo, vendar ne na vseh področjih. Rezultati kažejo tudi razlike v implementacijah posameznih rešitev. LXC in Docker sta menjavala svoji mesti pri različnih testih, ravno tako KVM in Xen. Vse to skupaj pomeni, da ne moremo kar brez zadržkov določiti področja (npr. računsko prevladujoče aplikacije), na katerem je "dobra" le ena tehnologija. Pri takih rezultatih se je tako pametneje osredotočiti na izbiro konkretne rešitve namesto izbiro tehnologije.

Gledati moramo širše. Vsebniki so zagotovo dobro orodje za razvoj in distribucijo aplikacij. V vsebnik zapakirana aplikacija je samozadostna, izognemo se težavam

z neustreznimi verzijami knjižnic, dovoljenji za dostop, morebitne spremembe na gostiteljskemu sistemu (ki ne spreminjajo vsebnikovega "hipervizorja") nam ne bodo povzročile težav ipd. Manjši stroški režije pomenijo tudi, da lahko en gostiteljski sistem uporabimo za streženje več aplikacij naenkrat. Če se omejimo le na storitve v oblaku, potem vidimo pogostejšo uporabo vsebnikov v storitvah vrste PaaS in SaaS. Storitve vrste IaaS pa bodo najverjetneje ostale večinoma domena virtualizacije. Uporabniki takih sistemov želijo nadzor nad navideznim strojem, ki ga "najemajo", to pa pomeni tudi lastno izbiro operacijskega sistema (izbiro verzije in arhitekture).
Prihodnost pa vidimo tudi v kombinirani uporabi, pri čemer v navideznih strojih poganjamo vsebnike, primer so npr. na IaaS postavljene storitve PaaS in SaaS.

Testiranja smo opravljali s poganjanjem enega navideznega stroja/vsebnika naenkrat. Nadaljne delo na tem področju bi bilo opravljanje testiranj, pri čemer sočasno teče več navideznih strojev/vsebnikov. Zagotovo v takem primeru skupni stroški režije pri virtualizaciji bolj obremenijo gostitelja, kar pomeni, da se lahko skupna prednost tehnologije vsebnikov poveča. Drugo področje bi bilo tudi testiranje na drugačni strojni opremi. Naš testni sistem je bil namizni računalnik z enim večjedrnim procesorjem, rezultati na pravem večprocesorskem sistemu so lahko drugačni. Vpliv topologije sistema smo lahko videli pri testiranju v Xenovem navideznem stroju vrste PVH, ki v navidezni stroj ni izpostavil ustreznih podatkov o procesorju. Pri aplikacijah, ki se dinamično prilagajajo strojni opremi zaradi optimizacije, lahko tako pričakujemo razlike v performančnih zmogljivostih.
Razvoj obeh tehnologij je zelo aktiven, izboljšave in nove funkcionalnosti prihajajo z vsako novo izdajo, izboljšave na področju podpore virtualizaciji pa se pojavljajo tudi pri procesorjih. Vse to pomeni, da bodo taka primerjalna testiranja aktualna tudi v prihodnje.

Poglavje 7

Sklepne ugotovitve

V tem delu smo najprej predstavili storitve v oblaku ter vpliv virtualizacije in tehnologije vsebnikov na delovanje in karakteristike teh storitev. Nato smo proučili obe tehnologiji in ju primerjali med seboj. Pri virtualizaciji smo predstavili problematiko virtualizacije arhitekture x86 in izvedbe rešitev te problematike (polna virtualizacija, strojno podprta virtualizacija in paravirtualizacija), pri tehnologiji vsebnikov pa pregledali delovanje te tehnologije v operacijskem sistemu Linux.
Zanimale so nas razlike v performančnih zmogljivostih obeh tehnologij, saj naj bi tehnologija vsebnikov imela prednost zaradi manjših stroškov režije. Zanimalo nas je, kolikšna in na katerih področjih je ta razlika v zmogljivostih. Postavili smo testni poligon in najprej opravili testiranje na fizičnem sistemu (za referenčne vrednosti), nato pa še na izbranih rešitvah za virtualizacijo oz. tehnologijo vsebnikov: KVM, Xen, LXC in Docker. Izvajali smo teste, ki so pokrivali različna področja: Intel Optimized LINPACK, STREAM, RandomAccess, netperf, fio, bonnie++ in UnixBench.
Analizirali smo rezultate testiranj in ugotovili, da ima v splošnem tehnologija vsebnikov pred virtualizacijo rahlo prednost, vendar ne na vseh področjih. Poleg izenačenosti tehnologij na določenih področjih so testi pokazali tudi razlike med rešitvami iste tehnologije. Na podlagi teh rezultatov smo sklenili, da se moramo pri iskanju primernih rešitev za konkretno področje osredotočiti na izbiro konkretne rešitve, in ne le na izbiro tehnologije.

Literatura

[1] K. Agarwal, B. Jain, D. E. Porter, "Containing the hype" v zborniku Proceedings of the 6th Asia-Pacific Workshop on Systems, ACM, 2015, str. 8.

[2] O. Agesen et al, "The evolution of an x86 virtual machine monitor", ACM SIGOPS Operating Systems Review, 2010, vol. 44., št.4: str. 3–18.

[3] P. Barham et al, "Xen and the art of virtualization." ACM SIGOPS Operating Systems Review, 2003, vol. 37., št. 5: str. 164–177.

[4] B. Bitner, S. Greenlee, "z/VM – A Brief Review of Its 40 Year History". Dosegljivo: `http://www.vm.ibm.com/vm40hist.pdf` (pridobljeno: 26. 4. 2016)

[5] O. Berghmans, "Nesting virtual machines in virtualization test frameworks", 2010. Dosegljivo: `http://www.math.technion.ac.il/~nyh/nested/Thesis_OlivierBerghmans_FinalVersion.pdf` (pridobljeno 25. 4. 2016)

[6] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes", IEEE Cloud Computing, 2014, št. 3: str. 81–84.

[7] E. Bugnion et al, "Bringing virtualization to the x86 architecture with the original VMware Workstation", ACM Transactions on Computer Systems (TOCS), 2012, vol. 30., št.4: str. 12:1–12:51.

[8] M. Carroll, A. Van der Merwe, P. Ketze, "Secure cloud computing: Benefits, risks and controls", v zborniku Information Security South Africa (ISSA), 2011, str. 1–9.

[9] N. Elhage, "Virtunoid: Breaking out of KVM", Black Hat USA, 2011. Dosegljivo: http://media.blackhat.com/bh-us-11/Elhage/BH_US_11_Elhage_Virtunoid_Slides.pdf (pridobljeno: 1. 5. 2016)

[10] W. Felter et al, "An updated performance comparison of virtual machines and linux containers", v zborniku 2015 IEEE International Symposium On Performance Analysis of Systems and Software (ISPASS), 2015. str. 171–172.

[11] C. Gong et al, "The characteristics of cloud computing", v zborniku 39th International Conference on Parallel Processing Workshops (ICPPW), 2010, str. 275–279

[12] IBM, "Google and IBM Announced University Initiative to Address Internet-Scale Computing Challenge". Dosegljivo: http://www-03.ibm.com/press/us/en/pressrelease/22414.wss (pridobljeno: 27. 4. 2016)

[13] A.M. Jey, "Performance comparison between linux containers and virtual machines" v zborniku 2015 International Conference on Advances in Computer Engineering and Applications (ICACEA), 2015, str. 342–346.

[14] A. Kivity et al, "kvm: the Linux virtual machine monitor", v zborniku Proceedings of the Linux symposium, 2007, str. 225–230.

[15] W. Li, A. Kanso, A. Gherbi, "Leveraging linux containers to achieve high availability for cloud services", v zborniku 2015 IEEE International Conference on Cloud Engineering (IC2E), 2015, str. 76–83.

[16] S. Lohr, "Google and IBM join in 'cloud computing' research". New York Times, 2007, 8. Dosegljivo: http://extrememediastudies.org/extreme_media/4_reading/pdf/CloudComputing_times.pdf (pridobljeno: 27. 5. 2015)

[17] P. Mell, T. Grance, "The NIST Definition of Cloud Computing", 2001. Dosegljivo: http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf (pridobljeno: 27. 4. 2016)

[18] M. B. Mollah, R. Kazi, S. Sikder, "Next generation of computing through cloud computing technology", v zborniku 25th IEEE Canadian Conference on Electrical & Computer Engineering (CCECE), 2012, str. 1–6.

[19] R. Morabito, J. Kjällman, M. Komu, Miika, "Hypervisors vs. lightweight virtualization: a performance comparison", v zborniku IEEE International Conference on Cloud Engineering (IC2E), 2015, str. 386–393.

[20] S. Pearson, A. Benameur, "Privacy, security and trust issues arising from cloud computing", v zborniku IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom), 2010, str. 693–702.

[21] D. M. Pham, "Performance comparison between x86 virtualization technologies", 2014. Dosegljivo: `http://gradworks.umi.com/1528024.pdf` (pridobljeno: 28. 2. 2016)

[22] I. Pratt et al, "Xen 3.0 and the art of virtualization". V Linux symposium. Ottawa, Ontario, Canada, 2005, str. 65–78.

[23] VMware information guide, "Software and Hardware Techniques for x86 Virtualization". Dosegljivo: `https://www.vmware.com/files/pdf/software_hardware_tech_x86_virt.pdf` (pridobljeno: 8. 3. 2016)

[24] VMware white paper "Virtualization Overview". Dosegljivo: `https://www.vmware.com/pdf/virtualization.pdf` (pridobljeno: 29. 7. 2015)

[25] VMware white paper "Virtualization Overview". Dosegljivo: `http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf` (pridobljeno: 29. 7. 2015)

[26] W. Guohui, T.S.E. NG, "The impact of virtualization on network performance of amazon ec2 data center", v zborniku INFOCOM, 2010 Proceedings IEEE, 2010, str. 1–9.

[27] J. Yang, Z. Chen, "Cloud computing research and security issues" v zborniku International Conference on Computational Intelligence and Software Engineering (CiSE), 2010, str. 1–3.

[28] M. G. Xavier et al, "Performance evaluation of container-based virtualization for high performance computing environments" v zborniku 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, IEEE, 2013. str. 233–240.

Ostali viri

[29] Amazon Web Services. Dosegljivo: https://aws.amazon.com/ (pridobljeno: 26. 4. 2016)

[30] Bonnie++. Dosegljivo: http://www.coker.com.au/bonnie++/ (pridobljeno 26. 4. 2016)

[31] CRIU - Checkpoint/Restore In Userspace. Dosegljivo: http://criu.org/Main_Page (pridobljeno 1. 6. 2016)

[32] Docker. Dosegljivo: https://www.docker.com (pridobljeno: 30. 4. 2016)

[33] fio - flexible I/O tester, Dosegljivo: http://linux.die.net/man/1/fio (pridobljeno 30. 4. 2016)

[34] FreeBSD Handbook, Jails. Dosegljivo: https://www.freebsd.org/doc/handbook/jails.html (pridobljeno: 30. 4. 2016)

[35] HPC Challenge. Dosegljivo: http://icl.cs.utk.edu/hpcc/ (pridobljeno 30. 5. 2016)

[36] IBM, developerWorks: RESTful Web services: The basics. Dosegljivo: http://www.ibm.com/developerworks/library/ws-restful/ (pridobljeno 30. 5. 2016)

[37] IBM, "Exploiting IBM AIX Workload Partitions". Dosegljivo: http://www.redbooks.ibm.com/redbooks/pdfs/sg247955.pdf (pridobljeno 30. 4. 2016)

[38] Intel Math Kernel Library. Dosegljivo: https://software.intel.com/en-us/intel-mkl (pridobljeno 30. 4. 2016)

[39] Intel Math Kernel Library Benchmarks, paket za Linux. Dosegljivo: http://registrationcenter-download.intel.com/akdlm/irc_nas/9128/l_mklb_p_11.3.3.011.tgz (pridobljeno 30. 4. 2016)

[40] Kernel Virtual Machine. Dosegljivo: http://www.linux-kvm.org (pridobljeno 30. 4. 2016)

[41] Linux Manual, getpid. Dosegljivo: http://man7.org/linux/man-pages/man2/getpid.2.html (pridobljeno: 30. 5. 2016)

[42] Linux Manual, stat. Dosegljivo: http://linux.die.net/man/1/stat (pridobljeno: 30. 4. 2016)

[43] LXC. Dosegljivo: https://linuxcontainers.org/lxc/introduction/ (pridobljeno: 29. 4. 2016)

[44] LXC Releases. Dosegljivo: https://github.com/lxc/lxc/releases (pridobljeno: 30. 4. 2016)

[45] Linux Programmer's Manual, Namespaces. Dosegljivo: http://man7.org/linux/man-pages/man7/namespaces.7.html (pridobljeno: 29. 4. 2016)

[46] Linux-VServer. Dosegljivo: http://linux-vserver.org (pridobljeno 30. 4. 2016)

[47] J.D. McCalpin, "STREAM: Sustainable Memory Bandwidth in High Performance Computers", 1991–2007. Dosegljivo: http://www.cs.virginia.edu/stream/ (pridobljeno: 27. 4. 2016)

[48] Microsoft Azure. Dosegljivo: https://azure.microsoft.com/en-us/ (pridobljeno 30. 4. 2016)

[49] Netperf. Dosegljivo: http://www.netperf.org/netperf/ (pridobljeno 30. 4. 2016)

[50] OpenVZ. Dosegljivo: https://openvz.org (pridobljeno 30. 4. 2016)

[51] QEMU. Dosegljivo: http://qemu.org (pridobljeno 1. 5. 2016)

[52] Oracle Enterprise Manager Ops Center User's Guide, Oracle Solaris Zones. Dosegljivo: `https://docs.oracle.com/cd/E18440_01/doc.111/e18415/chapter_zones.htm` (pridobljeno: 30. 4. 2016)

[53] Oracle VM, User's Guide for Release 3.2. Dosegljivo: `https://docs.oracle.com/cd/E35328_01/E35332/html/index.html` (pridobljeno: 1. 5. 2016)

[54] Red Hat Enterprise Linux 6, Resource Management Guide, Introduction to Control Groups. Dosegljivo: `https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Resource_Management_Guide/ch01.html` (pridobljeno: 29. 4. 2016)

[55] UnixBench. Dosegljivo: `https://github.com/kdlucas/byte-unixbench` (pridobljeno 30. 4. 2016)

[56] VirtualBox. Dosegljivo: `https://www.virtualbox.org/` (pridobljeno: 28. 4. 2016)

[57] VirtualBox Guest Additions. Dosegljivo: `https://www.virtualbox.org/manual/ch04.html` (pridobljeno: 28. 4. 2016)

[58] VMware. Dosegljivo: `https://www.vmware.com` (pridobljeno: 28. 4. 2016)

[59] VMware, "Overview of VMware Tools". Dosegljivo: `https://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=340` (pridobljeno: 28. 4. 2016)

[60] The Xen Project. Dosegljivo: `http://www.xenproject.org/` (pridobljeno: 1. 5. 2016)

[61] XenServer. Dosegljivo: `http://xenserver.org/` (pridobljeno: 1. 5. 2016)

Dodatek A

Nastavitve

A.1 Konfiguracija navideznega stroja KVM

```
<domain type='kvm'>
  <name>kvm_ubuntu16.04_amd64</name>
  <uuid>1c603d23-23ed-4111-9cc3-77375f81afde</uuid>
  <memory unit='KiB'>8388608</memory>
  <currentMemory unit='KiB'>8388608</currentMemory>
  <vcpu placement='static'>4</vcpu>
  <os>
    <type arch='x86_64' machine='pc-i440fx-wily'>hvm</type>
    <bootmenu enable='yes'/>
  </os>
  <features>
    <acpi/>
    <apic/>
  </features>
  <cpu mode='host-model'>
    <model fallback='allow'/>
  </cpu>
  <clock offset='utc'>
    <timer name='rtc' tickpolicy='catchup'/>
    <timer name='pit' tickpolicy='delay'/>
    <timer name='hpet' present='no'/>
  </clock>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>restart</on_crash>
  <pm>
    <suspend-to-mem enabled='no'/>
    <suspend-to-disk enabled='no'/>
  </pm>
  <devices>
    <emulator>/usr/bin/kvm-spice</emulator>
    <disk type='file' device='cdrom'>
      <driver name='qemu' type='raw'/>
      <target dev='hda' bus='ide'/>
      <readonly/>
      <address type='drive' controller='0' bus='0' target='0' unit='0'/>
    </disk>
    <disk type='file' device='disk'>
```

```
    <driver name='qemu' type='raw'/>
    <source file='/home/vm/kvm_ubuntu16.04_amd64.raw'/>
    <target dev='vda' bus='virtio'/>
    <boot order='1'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x07' function='0x0'/>
  </disk>
  <disk type='block' device='disk'>
    <driver name='qemu' type='raw'/>
    <source dev='/dev/sdd'/>
    <target dev='vdb' bus='virtio'/>
  </disk>
  <controller type='usb' index='0' model='ich9-ehci1'>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x7'/>
  </controller>
  <controller type='usb' index='0' model='ich9-uhci1'>
    <master startport='0'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x0' multifunction='on'/>
  </controller>
  <controller type='usb' index='0' model='ich9-uhci2'>
    <master startport='2'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x1'/>
  </controller>
  <controller type='usb' index='0' model='ich9-uhci3'>
    <master startport='4'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x2'/>
  </controller>
  <controller type='pci' index='0' model='pci-root'/>
  <controller type='ide' index='0'>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x1'/>
  </controller>
  <controller type='virtio-serial' index='0'>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0'/>
  </controller>
  <interface type='bridge'>
    <mac address='52:54:00:ff:f8:21'/>
    <source bridge='br0'/>
    <model type='virtio'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
  </interface>
  <serial type='pty'>
    <target port='0'/>
  </serial>
  <console type='pty'>
    <target type='serial' port='0'/>
  </console>
  <channel type='spicevmc'>
    <target type='virtio' name='com.redhat.spice.0'/>
    <address type='virtio-serial' controller='0' bus='0' port='1'/>
  </channel>
  <input type='mouse' bus='ps2'/>
  <input type='keyboard' bus='ps2'/>
  <graphics type='vnc' port='-1' autoport='yes'/>
  <sound model='ich6'>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x04' function='0x0'/>
  </sound>
  <video>
    <model type='qxl' ram='65536' vram='65536' vgamem='16384' heads='1'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x02' function='0x0'/>
  </video>
  <redirdev bus='usb' type='spicevmc'>
  </redirdev>
  <redirdev bus='usb' type='spicevmc'>
  </redirdev>
  <memballoon model='virtio'>
```

```
      <address type='pci' domain='0x0000' bus='0x00' slot='0x08' function='0x0'/>
    </memballoon>
  </devices>
</domain>
```

A.2 Konfiguracija navideznega stroja Xen HVM

```
name = "ubuntu64"
memory = 8192
vcpus = 4
builder = "hvm"
hap = 1
vif = [ 'script=vif-bridge,bridge=br0' ]
disk = [ 'file:/home/vm/xen_ubuntu16.04_amd64.raw,xvda,w', 'phy:/dev/sdd,xvdb,w' ]
```

A.3 Konfiguracija navideznega stroja Xen PVH

```
name = "ubuntu64"
#kernel = "/home/vm/net_install_amd64/vmlinuz"
#ramdisk = "/home/vm/net_install_amd64/initrd.gz"
bootloader = "/usr/lib/xen-4.6/bin/pygrub"
memory = 8192
vcpus = 4
#popravek, da navidezni stroj pravilno vidi 4 jedra
cpuid = ['0x1:ebx=xxxxxxxx00000001xxxxxxxxxxxxxxxx']
pvh = 1
vif = [ 'script=vif-bridge,bridge=br0' ]
disk = [ 'file:/home/vm/xen_ubuntu16.04_amd64.raw,xvda,w', 'phy:/dev/sdd,xvdb,w' ]
```

A.4 Konfiguracija vsebnika LXC

Datoteka lxc_config

```
# Distribution configuration
lxc.include = /usr/share/lxc/config/ubuntu.common.conf
lxc.arch = x86_64

# Container specific configuration
lxc.rootfs = /home/vm/containers/ubuntu_16.04/rootfs
lxc.rootfs.backend = dir
lxc.utsname = ubuntu_16.04

# Network configuration
lxc.network.type = veth
lxc.network.link = br0
lxc.network.flags = up
lxc.network.hwaddr = 00:16:3e:4e:c4:00

lxc.cgroup.memory.limit_in_bytes = 8192M

lxc.mount = /home/vm/containers/ubuntu_16.04/fstab
```

Datoteka fstab

```
/test test none bind,create=dir 0 0
```

A.5 Konfiguracija vsebnika Docker

Zagon vsebnika

```
docker run -ti -m 8192M --net=none -p 9000 -p 12865-13000 -v /test:/test test /bin/bash
```

Nastavitev omrežja

```
pid=`docker inspect -f '{{.State.Pid}}' $1`
mkdir -p /var/run/netns
ln -s /proc/$pid/ns/net /var/run/netns/$pid
ip link add A type veth peer name B
brctl addif br0 A
ip link set A up
ip link set B netns $pid
ip netns exec $pid ip link set dev B name eth0
ip netns exec $pid ip link set eth0 address 12:34:56:78:9a:bc
ip netns exec $pid ip link set eth0 up
ip netns exec $pid ip addr add 10.0.10.40/8 dev eth0
ip netns exec $pid ip route add default via 10.0.10.1
```

A.6 Konfiguracija testov

A.6.1 Intel Optimized LINPACK Benchmark - vnosna datoteka

```
Intel(R) Optimized LINPACK Benchmark data file
Intel(R) Optimized LINPACK Benchmark data
1     # number of tests
31000 # problem sizes
31000 # leading dimensions
10    # times (trials) to run a test
1     # alignment values (in KBytes)
```

A.6.2 STREAM - Makefile

```
CC = gcc
CFLAGS = -O2

FF = g77
FFLAGS = -O2

all: stream_f.exe stream_c.exe

stream_f.exe: stream.f mysecond.o
	$(CC) $(CFLAGS) -c mysecond.c
	$(FF) $(FFLAGS) -c stream.f
	$(FF) $(FFLAGS) stream.o mysecond.o -o stream_f.exe

stream_c.exe: stream.c
	$(CC) $(CFLAGS) stream.c -o stream_c.exe

clean:
	rm -f stream_f.exe stream_c.exe *.o

test: stream.c
	$(CC) $(CFLAGS) -fopenmp -DSTREAM_ARRAY_SIZE=100000000 stream.c -o test_stream
```

A.6.3 HPCC (RandomAccess) - vnosna datoteka

```
HPLinpack benchmark input file
Innovative Computing Laboratory, University of Tennessee
HPL.out      output file name (if any)
8            device out (6=stdout,7=stderr,file)
1            # of problems sizes (N)
1000         Ns
1            # of NBs
80           NBs
0            PMAP process mapping (0=Row-,1=Column-major)
1            # of process grids (P x Q)
1            Ps
1            Qs
16.0         threshold
1            # of panel fact
2            PFACTs (0=left, 1=Crout, 2=Right)
1            # of recursive stopping criterium
4            NBMINs (>= 1)
1            # of panels in recursion
2            NDIVs
1            # of recursive panel fact.
1            RFACTs (0=left, 1=Crout, 2=Right)
1            # of broadcast
1            BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
1            # of lookahead depth
1            DEPTHs (>=0)
2            SWAP (0=bin-exch,1=long,2=mix)
64           swapping threshold
0            L1 in (0=transposed,1=no-transposed) form
0            U  in (0=transposed,1=no-transposed) form
1            Equilibration (0=no,1=yes)
8            memory alignment in double (> 0)
###### This line (no. 32) is ignored (it serves as a separator). ######
0                               Number of additional problem sizes for PTRANS
1200 10000 30000                values of N
0                               number of additional blocking sizes for PTRANS
40 9 8 13 13 20 16 32 64        values of NB
```

A.6.4 netperf - zagonski argumenti

```
netperf -l 60 -H 10.0.10.1 -t TCP_RR -i 10,3 -I 95,5 -- -r 1024K,2048K
netperf -l 60 -H 10.0.10.1 -t TCP_CRR -i 10,3 -I 95,5 -- -r 1024K,2048K
```

A.6.5 fio - nastavitve pošiljatelja

```
[sender]
hostname=10.0.10.1
port=9000
proto=tcp
rw=write
rate=200M
ioengine=net
size=30G
```

A.6.6 fio - nastavitve prejemnika

```
[receiver]
listen=1
proto=tcp
port=9000
rw=read
ioengine=net
size=30G
```

A.6.7 bonnie++ - zagonski argumenti

```
bonnie++ -s 32G -d /test/tmp -n 128:1k:1k -x 10
```

Dodatek B

Meritve

	HW	Docker	LXC	Xen(PVH)	Xen(HVM)	KVM
1	184,3373	184,4496	184,1743	183,2085	182,645	181,8257
2	184,4124	184,4158	184,2019	183,2249	182,7546	181,9809
3	184,4271	184,4073	184,2665	183,2362	182,7797	182,0334
4	184,4205	184,4032	184,2594	183,28	182,7628	182,1021
5	184,5255	184,5494	184,395	183,3744	182,8749	182,3333
6	185,1747	185,1787	184,8316	184,0471	183,5876	182,9863
7	185,1807	185,168	184,2503	184,0549	183,5776	182,9552
8	185,1924	185,1409	184,2814	184,0795	183,6905	182,9877
9	185,1819	185,1628	184,2769	184,0881	183,6107	183,0368
10	185,1830	185,185	184,298	184,0806	183,5915	182,8515

Tabela B.1: Intel MKL LINPACK - meritve (Gflops)

	Gflops	Std. dev.	Std. napaka
HW	184,8036	0,381375259	0,07%
Docker	184,8061	0,363233338	0,06%
LXC	184,3235	0,178223265	0,03%
Xen(PVH)	183,6674	0,404985696	0,07%
Xen(HVM)	183,1875	0,428214854	0,07%
KVM	182,5093	0,471207346	0,08%

Tabela B.2: Intel MKL LINPACK - povprečne vrednosti

	1	2	3	4	5	6	7	8	9	10
HW Copy	14802,2	14798,1	14792,1	14794,3	14800	14805	14792,3	14832,4	14804	14805,8
HW Scale	14870,3	14873,1	14857,4	14868,7	14857,6	14873,6	14868,6	14872,7	14863,3	14873,1
HW Add	16907,7	16900,2	16912,5	16921,9	16911,4	16917,6	16907,7	16899,1	16905,6	16914,2
HW Triad	16859,0	16851,4	16863,9	16863,6	16862,5	16862,2	16854,6	16858,4	16857,5	16847,7
Docker Copy	14821	14798,8	14799,3	14802,1	14798,2	14811,3	14794,6	14793,9	14799,7	14817,3
Docker Scale	14879,1	14860,4	14873,6	14860,1	14871,3	14867,6	14858,8	14866	14865,3	14861,3
Docker Add	16904,8	16902,2	16916,5	16894,3	16907,1	16915,7	16893,1	16897,2	16906,3	16908,5
Docker Triad	16860,1	16860,7	16855,7	16851,2	16857,4	16851,8	16851,9	16855,8	16853,6	16846,0
LXC Copy	14437,6	14445,9	14431,3	14435,3	14442,8	14446,3	14436,3	14437,8	14447,5	14438,6
LXC Scale	14514,1	14516,3	14500,9	14502,9	14503,9	14510,9	14504,7	14507,3	14516,2	14506,3
LXC Add	16492	16502	16495,3	16493,8	16485,3	16496,5	16512,7	16495,3	16498,5	16496,4
LXC Triad	16459,9	16443,6	16442,3	16453,6	16448,6	16447,6	16452,3	16455,9	16450,9	16444,8
KVM Copy	14746	14730,3	14728,4	14736	14739,4	14736,2	14739,4	14735,8	14742,3	14743,6
KVM Scale	14826,3	14800,7	14815,2	14817,3	14810,4	14803,4	14813	14807,1	14816,1	14823,9
KVM Add	16862,9	16833,4	16832,8	16829,2	16820,4	16832,1	16829,9	16835,1	16832,3	16824,6
KVM Triad	16782,3	16753,1	16739,3	16763,3	16761,3	16745,6	16762,0	16774,7	16788,0	16738,4
Xen(HVM) Copy	14695,6	14666,1	14659,5	14686	14686,1	14666,1	14677	14660,2	14682,8	14666,4
Xen(HVM) Scale	14779,2	14738,5	14754,6	14756,1	14764,6	14750,5	14764	14763,3	14769	14744,1
Xen(HVM) Add	16768,6	16716,9	16715,4	16691,6	16744,2	16698,8	16752,8	16691,2	16759	16723,8
Xen(HVM) Triad	16693,7	16604,6	16615,0	16553,4	16682,9	16624,2	16680,8	16611,6	16671,3	16643,5
Xen(PVH) Copy	14725,2	14711,5	14714,1	14705,5	14711,1	14741,5	14712,1	14711,4	14713	14711
Xen(PVH) Scale	14792,3	14801,8	14809,1	14800,8	14795	14809,7	14798,5	14800,4	14804,9	14808,4
Xen(PVH) Add	16813	16797,7	16799,5	16799,3	16795,1	16815,1	16787,1	16809,7	16801,6	16816,5
Xen(PVH) Triad	16744,6	16745,1	16741,2	16730,3	16735,3	16773,2	16725,6	16739,2	16740,6	16742,2

Tabela B.3: STREAM - meritve (MB/s)

	MB/s	Std. dev.	Std. napaka
HW Copy	14802,6	11,63517655	0,02%
HW Scale	14867,8	6,271310334	0,01%
HW Add	16909,8	7,223180273	0,01%
HW Triad	16858,1	5,442793407	0,01%
Docker Copy	14803,6	9,504244668	0,02%
Docker Scale	14866,4	6,661539697	0,01%
Docker Add	16904,6	8,083323026	0,02%
Docker Triad	16854,4	4,463132931	0,01%
LXC Copy	14439,9	5,40148127	0,01%
LXC Scale	14508,4	5,657099768	0,01%
LXC Add	16496,8	7,085007649	0,01%
LXC Triad	16450,0	5,645696491	0,01%
KVM Copy	14737,7	5,583945241	0,01%
KVM Scale	14813,3	8,249606052	0,02%
KVM Add	16833,3	11,31646293	0,02%
KVM Triad	16760,8	17,1839072	0,03%
Xen(HVM) Copy	14674,6	12,57562015	0,03%
Xen(HVM) Scale	14758,4	12,11724666	0,03%
Xen(HVM) Add	16726,2	28,45456613	0,05%
Xen(HVM) Triad	16638,1	44,44534721	0,08%
Xen(PVH) Copy	14715,6	10,33915965	0,02%
Xen(PVH) Scale	14802,1	5,947632586	0,01%
Xen(PVH) Add	16803,5	9,676110324	0,02%
Xen(PVH) Triad	16741,7	12,69427255	0,02%

Tabela B.4: STREAM - povprečne vrednosti

	HW	Docker	KVM	LXC	Xen(HVM)
1	0,456892	0,41593	0,4166	0,408228	0,38047
2	0,454719	0,442816	0,414871	0,440929	0,422909
3	0,443284	0,457201	0,44115	0,458608	0,447251
4	0,436552	0,410802	0,400368	0,440752	0,150248
5	0,455898	0,45266	0,365301	0,455803	0,433476
6	0,454719	0,419341	0,427472	0,432219	0,42843
7	0,447935	0,448094	0,398248	0,458392	0,434912
8	0,440311	0,420866	0,427826	0,434654	0,420283
9	0,408873	0,433028	0,359906	0,409121	0,442059
10	0,452381	0,431604	0,450712	0,427036	0,433476

Tabela B.5: RandomAccess meritve (GUPS)
(rdeča barva označuje izločeno meritev)

	GUPS	Std. dev.	Std. napaka
HW	0,4451564	0,013832617	0,98%
LXC	0,4365742	0,017431257	1,26%
Docker	0,4332342	0,015542606	1,13%
Xen(HVM)	0,427029556	0,018294765	1,35%
KVM	0,4102454	0,028382751	2,19%

Tabela B.6: RandomAccess - povprečne vrednosti

	HW	Docker	LXC	KVM	Xen(HVM)	Xen(PVH)
1	37,13	37,04	37,06	37,09	36,8	36,82
2	37,14	37,02	37,09	37,09	36,8	36,8
3	37,14	37,05	37,09	37,04	36,79	36,8
4	37,14	37,04	37,09	37,1	36,79	36,8
5	37,1	37,03	37,08	37,09	36,79	36,8
6	36,98	37,04	37,08	37,1	36,79	36,8
7	37,13	37,04	37,08	37,1	36,8	36,8
8	37,13	37,03	37,09	37,1	36,8	36,8
9	37,14	37,04	37,08	37,09	36,8	36,8
10	37,14	37,04	37,08	37,1	36,78	36,8

Tabela B.7: NetPerf, zahteva/odgovor - meritve (št.prenosov/s)

	HW	Docker	LXC	KVM	Xen(HVM)	Xen(PVH)
1	36,89	36,87	36,89	36,86	35,25	35,13
2	36,9	36,87	36,89	36,87	35,28	35,27
3	36,9	36,87	36,88	36,87	35,25	35,25
4	36,93	36,87	36,87	36,85	35,24	35,23
5	36,89	36,87	36,88	36,85	35,29	35,08
6	36,92	36,87	36,87	36,86	35,29	35,04
7	36,92	36,87	36,88	36,86	35,28	35,24
8	36,9	36,87	36,87	36,86	35,29	35,26
9	36,92	36,87	36,88	36,87	35,24	35,28
10	36,91	36,87	36,88	36,87	35,24	35,29

Tabela B.8: NetPerf, povezava/zahteva/odgovor - meritve (št.prenosov/s)

	št.prenosov/s	Std. dev	Std. napaka
HW	37,117	0,049676733	0,04%
Docker	37,037	0,008232726	0,01%
LXC	37,082	0,009189366	0,01%
KVM	37,09	0,018257419	0,02%
Xen(HVM)	36,794	0,006992059	0,01%
Xen(PVH)	36,802	0,006324555	0,01%

Tabela B.9: NetPerf, zahteva/odgovor
povpr. vrednosti

	št.prenosov/s	Std. dev	Std. napaka
HW	36,908	0,013984118	0,01%
Docker	36,87	0	0,00%
LXC	36,879	0,007378648	0,01%
KVM	36,862	0,007888106	0,01%
Xen(HVM)	35,265	0,022730303	0,02%
Xen(PVH)	35,207	0,089697021	0,08%

Tabela B.10: NetPerf, povezava/zahteva/odgovor
povprečne vrednosti

	HW		Docker		LXC		KVM		Xen(HVM)		Xen(PVH)	
	največja	povprečje	največja	povprečje	največja	povprečje	največja	povprečje	največja	povprečje	največja	povprečje
1	919872	917897,83	919872	917649,99	919872	917889,55	919680	917611,22	919872	917622,19	920064	917611,33
2	919872	917866,83	919872	917847,82	919872	917859,98	919680	917584,58	2758080	919985,98	919872	917622,47
3	919872	917885,69	919872	917864,1	919872	917884,15	919680	917602,68	919744	917647,24	919808	917624,53
4	919872	917863,95	919872	917870,3	919872	917614,97	919680	917520,84	919744	917493,95	919936	917481,82
5	919872	917874,63	919872	917879,11	919872	917880,78	919680	917597,12	919744	917642,75	919872	917589,87
6	919872	917888,39	919872	917854,79	919872	917871,34	919680	917621,28	2758208	922333,95	919872	917508,66
7	919872	917888,68	919872	917834,23	919872	917863,84	919680	917606,3	919808	917610,08	919808	917612,66
8	919872	917886,59	919872	917819,49	919872	917647,42	919680	917582,9	2757568	920083,98	919808	917627,63
9	919872	917892,28	919872	917861,64	919872	917866,18	919680	917592,73	919744	917640,94	2758272	919774,06
10	919872	917878,58	919872	917840,05	919872	917777,49	919680	917618,33	919680	917612,15	919872	917496,66

Tabela B.11: Fio - meritve (Kbit/s)
(rdeča barva označuje izločeno meritev)

	št.prenosov/s	Std. dev	Std. napaka
HW	917882,345	11,04113747	0,00%
Docker	917832,152	66,43168875	0,00%
LXC	917815,57	102,3975786	0,00%
KVM	917593,798	28,74133562	0,00%
Xen(HVM)	917609,9	53,28761723	0,00%
Xen(PVH)	917575,07	60,88556782	0,00%

Tabela B.12: Fio - povprečje (Kbit/s)

	Sekv.pisanje (KB/s)	Sekv.branje (KB/s)	Naklj.kreir. (dat./s)	Naklj.prev. (dat./s)
HW 1	341161	670772	42588	215048
HW 2	329070	670265	41611	222015
HW 3	328606	671133	42578	224391
HW 4	329909	670049	41857	198637
HW 5	330080	669958	41222	242587
HW 6	327839	671495	42755	253993
HW 7	329670	670191	42466	234597
HW 8	329520	670434	42977	247635
HW 9	325414	669688	42669	252162
HW 10	330344	670374	42386	234514
Docker 1	339261	670317	41389	174176
Docker 2	329797	670095	40105	175240
Docker 3	329168	662567	41054	175478
Docker 4	329455	663101	40493	201035
Docker 5	329116	662214	40927	180280
Docker 6	330304	664360	40474	190022
Docker 7	329860	663736	40940	196497
Docker 8	328888	664327	41025	211463
Docker 9	328434	664451	40629	178176
Docker 10	329795	664104	40681	199550
LXC 1	339274	587637	40901	171692
LXC 2	330149	586185	40761	169052
LXC 3	328503	586816	41550	174924
LXC 4	329229	586471	41045	186149
LXC 5	328985	585809	40774	171038
LXC 6	330136	586691	40348	175754
LXC 7	329282	585763	40334	187947
LXC 8	329008	586872	40906	170489
LXC 9	328907	588820	40605	170281
LXC 10	327943	586317	40994	206126
KVM 1	339546	551939	35862	212565
KVM 2	326138	579768	34897	193559
KVM 3	329324	572576	35288	230512
KVM 4	329873	575394	35132	204454
KVM 5	327469	574342	35311	202576
KVM 6	329455	574480	35401	199643
KVM 7	328522	574823	36522	189149
KVM 8	328926	583421	35617	164513
KVM 9	325279	574124	34958	197196
KVM 10	329209	574535	35565	207126
Xen(HVM) 1	340246	523128	40698	203054
Xen(HVM) 2	328768	524795	40922	201014
Xen(HVM) 3	330530	520231	40661	207957
Xen(HVM) 4	330826	526334	40544	208460
Xen(HVM) 5	329283	526925	41381	216398
Xen(HVM) 6	330418	524007	45704	196006
Xen(HVM) 7	328772	528570	43794	223246
Xen(HVM) 8	328615	521570	42193	223421
Xen(HVM) 9	329700	536416	40967	219195
Xen(HVM) 10	329984	529807	42171	216001
Xen(PVH) 1	339932	584071	45824	204288
Xen(PVH) 2	329499	584766	39758	226565
Xen(PVH) 3	329717	582570	39467	214157
Xen(PVH) 4	330356	585450	40046	220956
Xen(PVH) 5	329293	584266	39514	218763
Xen(PVH) 6	330714	580886	39686	218766
Xen(PVH) 7	329477	586406	40743	222952
Xen(PVH) 8	328803	583180	40971	217853
Xen(PVH) 9	329968	582517	40152	217170
Xen(PVH) 10	329476	584205	40247	215654

Tabela B.13: Bonnie++ - meritve

	Povpr.	St.dev.	St.napaka
HW sekv.pisanje	330161,3	4127,004753	0,40%
HW sekv.branje	670435,9	552,8770107	0,03%
HW naklj.kreiranje	42310,9	560,6108672	0,42%
HW naklj.prev.	232557,9	17684,72457	2,40%
Xen(PVH) sekv.pisanje	330723,5	3280,574315	0,39%
Xen(PVH) sekv.branje	583831,7	1596,045464	0,03%
Xen(PVH) naklj.kreiranje	40640,8	1887,717422	0,44%
Xen(PVH) naklj.preverjanje	217712,4	5925,116693	2,57%
Docker sekv.pisanje	330407,8	3158,022722	0,39%
Docker sekv.branje	664927,2	2887,298691	0,03%
Docker naklj.kreiranje	40771,7	367,6855692	0,43%
Docker naklj.preverjanje	188191,7	13319,06125	2,97%
LXC sekv.pisanje	330141,6	3276,828216	0,40%
LXC sekv.branje	586738,1	915,1307678	0,03%
LXC naklj.kreiranje	40821,8	355,672696	0,43%
LXC naklj.preverjanje	178345,2	11804,23955	3,14%
KVM sekv.pisanje	329374,1	3880,882272	0,40%
KVM sekv.branje	573540,2	8245,510102	0,03%
KVM naklj.kreiranje	35455,3	478,0427805	0,50%
KVM naklj.preverjanje	200129,3	16972,18531	2,79%
Xen(HVM) sekv.pisanje	330714,2	3441,026617	0,33%
Xen(HVM) sekv.branje	526178,3	4676,408832	0,03%
Xen(HVM) naklj.kreiranje	41903,5	1671,054707	0,42%
Xen(HVM) naklj.preverjanje	211475,2	9584,041328	2,64%

Tabela B.14: Bonnie++ - povprečje

	1	2	3	4	5	6	7	8	9	10	Povp.	Std.dev.	Std.nap.
Dhrystone	12586,4	12611,3	12576,5	12593,1	12603,9	12595,6	12579,8	12598,2	12566	12545,3	12585,61	19,53836397	0,05%
Whetstone	1949	1949,1	1949	1949,1	1949	1949,1	1949,3	1949,3	1949,3	1949,3	1949,15	0,13540064	0,00%
Execl	5877,1	5878,8	5872,4	5884,1	5873,8	5876,9	5863,2	5854,4	5858,3	5871,2	5871,02	9,498514504	0,05%
Kopir.-1024B/2000B	3865,3	3784,1	3926,9	3585,2	3796,5	3767,9	3674,5	3819	3625,8	3878,6	3772,38	111,9565471	0,94%
Kopir.- 256B/500B	2324,9	2392,8	2462,2	2356,5	2246,8	2220,4	2252,6	2281,8	2365,6	2318,5	2322,21	74,6771637	1,02%
Kopir.-4096B/8000B	8117,9	7873,3	8054,8	8029,4	8105,7	7911,7	8029,1	8229,4	7844	8177,7	8037,3	128,3491938	0,50%
Uporaba cevi	8771,3	8771,7	8771,5	8746,7	8745,5	8754	8743,8	8754	8753	8745,7	8755,72	11,49258698	0,04%
Kom. med proc.	4270,3	4269,8	4275,9	4266,8	4264,2	4277	4262,2	4262,6	4285,3	4270,6	4270,47	7,269884608	0,05%
Kreir. procesa	5216,1	5215,8	5226,2	5271,8	5221,5	5249,3	5198	5248,8	5243,9	5207	5229,84	22,85379618	0,14%
Skripte - 1 sočasno	10775,4	10788,7	10776,5	10778,9	10793,7	10777,4	10800,9	10785,8	10762,9	10795,1	10783,53	11,40672803	0,03%
Skripte - 8 sočasno	9179,9	9164,4	9152	9150,3	9143,7	9163	9147,6	9138,8	9154,7	9164,7	9155,91	12,1757911	0,04%
Režija sist. klicev	6513,2	6531,6	6513,3	6513,5	6525,1	6520,3	6559,7	6512,7	6518,6	6534,6	6524,26	14,74623869	0,07%
Skupni indeks	5756,4	5747,6	5787,3	5724,5	5729,4	5712,9	5709,4	5746,3	5717,3	5757,8	5738,89	24,60101398	0,14%

Tabela B.15: UnixBench meritve - fizični računalnik

	1	2	3	4	5	6	7	8	9	10	Povp.	Std.dev.	Std.nap.
Dhrystone	12612,9	12618	12623,5	12583,3	12559,9	12604,5	12618,7	12486,2	12576,8	12522,3	12580,61	46,02625941	0,12%
Whetstone	1949,1	1949,1	1949,4	1949,2	1949,3	1949,3	1949,1	1949,1	1949	1949,3	1949,19	0,128668394	0,00%
Execl	5712,1	5719,5	5709,6	5719,6	5726,5	5719,3	5734,8	5712,6	5728,5	5718,7	5720,12	7,897369463	0,04%
Kopir.-1024B/2000B	3729,4	3735,9	3633	3715,3	3733,3	3636,5	3807,8	3759,4	3733,4	3767,7	3725,17	54,29360613	0,46%
Kopir.- 256B/500B	2402,5	2377,5	2367,9	2306,6	2341,4	2344,3	2336,9	2241,8	2311,8	2225,1	2325,58	56,56624042	0,77%
Kopir.-4096B/8000B	8055	8047,9	7927,4	8052,2	7790,6	7762,4	8085,5	7551,3	7749,2	7540,3	7856,18	208,0710018	0,84%
Uporaba cevi	6640,3	6653,1	6669,7	6653,5	6657,1	6649,1	6667,7	6662,5	6635	6650,9	6653,89	11,05184248	0,05%
Kom. med proc.	3476,3	3486	3487,1	3483	3486,2	3494,8	3485,5	3488,1	3477,5	3479,1	3484,36	5,571794046	0,05%
Kreir. procesa	5242,9	5236,2	5188,5	5207	5302,2	5173,6	5171,6	5225,3	5214	5173,5	5213,48	40,82713694	0,25%
Skripte - 1 sočasno	10594,5	10617	10583,5	10579,8	10589,6	10596,9	10582,6	10583,5	10585,8	10587,6	10590,08	10,87124648	0,03%
Skripte - 8 sočasno	9030,6	9037,2	9037	9048,6	9056,7	9041,2	9039,6	9028,7	9051,6	9054,9	9042,61	9,864577482	0,03%
Režija sist. klicev	5013	5021	5011,1	5019,4	5022,7	4991,9	5023	5001,2	5000,6	5042,2	5014,61	14,40520354	0,09%
Skupni indeks	5380,2	5380,2	5353,5	5359,2	5362,9	5337,5	5378,5	5319,6	5343,1	5317,1	5353,18	23,59127899	0,14%

Tabela B.16: UnixBench meritve - vsebnik LXC

	1	2	3	4	5	6	7	8	9	10	Povp.	Std.dev.	Std.nap.
Dhrystone	12619,3	12556,7	12589,5	12607,9	12552,9	12601,3	12609	12555,3	12546,2	12612,2	12585,03	28,90886254	0,07%
Whetstone	1949,3	1948,5	1949,2	1949	1948,4	1949,1	1949	1949	1948,9	1949,1	1948,95	0,287711275	0,00%
Execl	4488,2	4516,5	4506,8	4505,7	4499,8	4504,8	4470,3	4509,1	4501,3	4501,1	4500,36	12,82274022	0,09%
Kopir.-1024B/2000B	3703,1	3679,7	3756,6	3887,9	3736,6	3922,3	3757,7	3780,4	3732,5	3630,8	3758,76	88,6939081	0,75%
Kopir.- 256B/500B	2298,3	2305,4	2283,5	2383,4	2278,4	2285,6	2262,9	2344	2375,9	2382,2	2319,96	46,88025408	0,64%
Kopir.-4096B/8000B	7863,8	7757,4	7721,1	8055,2	7990,8	7914,1	7934,3	7858,7	7775,7	7865,1	7873,62	104,7603275	0,42%
Uporaba cevi	8530,9	8533,9	8542,2	8520,7	8527,7	8544,4	8545,8	8524,8	8528,9	8535,2	8533,45	8,497744801	0,03%
Kom. med proc.	3805,7	3801,8	3806,9	3801,8	3803,7	3802,1	3802,7	3800,2	3807,9	3803	3803,58	2,477812655	0,02%
Kreir. procesa	5225,7	5191,5	5236,2	5246,5	5231,9	5249,3	5204	5256,1	5204,9	5232,2	5227,83	21,42817719	0,13%
Skripte - 1 sočasno	9765,9	9822,5	9797,3	9806,2	9805,8	9802,2	9803,9	9795,8	9802	9792	9799,36	14,33730178	0,05%
Skripte - 8 sočasno	6910,9	6937,7	6922,7	6942,2	6922	6910,3	6936,7	6906,3	6908,6	6928	6922,54	13,29303911	0,06%
Režija sist. klicev	6341,4	6334,5	6306	6308,6	6314,7	6309,8	6326,4	6300,1	6289,5	6352,2	6318,32	19,72628252	0,10%
Skupni indeks	5339,1	5332,6	5336,5	5391,4	5345,7	5367,8	5341,3	5356,3	5347,2	5350,8	5350,87	17,54169952	0,10%

Tabela B.17: UnixBench meritve - vsebnik Docker

	1	2	3	4	5	6	7	8	9	10	Povp.	Std.dev.	Std.nap.
Dhrystone	12607,7	12560,2	12603,9	12608,8	12590,7	12624,2	12552	12518,7	12617	12606,2	12588,94	34,03593526	0,09%
Whetstone	1945,4	1945,2	1944,4	1944,5	1945,1	1945,3	1945,3	1945,1	1945,2	1944,8	1945,03	0,346570499	0,01%
Execl	4719,4	4725,9	4749,6	4746,2	4740,1	4765,8	4755,2	4754	4766,4	4763,1	4748,57	16,13340565	0,11%
Kopir.-1024B/2000B	3698	3741	3793	3701,3	3754	3712,3	3702,8	3773,2	3762,5	3728	3736,61	33,42718288	0,28%
Kopir.- 256B/500B	2361,3	2392,9	2361,1	2374,7	2361,2	2367,9	2367,7	2331	2331,3	2372,2	2362,13	18,83466367	0,25%
Kopir.-4096B/8000B	7923,2	7923,6	7639	7919,3	7793,6	7902,3	7838,9	7895,2	7826,2	7868,1	7852,94	87,55239955	0,35%
Uporaba cevi	8584	8563,9	8564	8580,1	8584,8	8561,5	8596	8567	8587,5	8560	8574,88	12,97697106	0,05%
Kom. med proc.	3436,4	3177,5	3266,2	3175,2	3408,6	3211	3156,3	3172,7	3451,6	3218,1	3267,36	118,2911211	1,14%
Kreir. procesa	3142,2	3883,6	3891,4	3626	3383	3891,8	3322,7	3303,2	3805,2	3931	3618,01	302,075408	2,64%
Skripte - 1 sočasno	9249,3	9274,2	9197,4	9248,5	9266,9	9240,8	9282,3	9255,5	9305,3	9322,1	9264,23	35,00873066	0,12%
Skripte - 8 sočasno	8241,5	8256,5	8274,7	8255,9	8266,2	8293,5	8293,9	8276,9	8298,3	8292,1	8274,95	19,56523277	0,07%
Režija sist. klicev	6333,4	6344,5	6340,3	6337,6	6340,7	6341,6	6339,8	6334,7	6354	6341,5	6340,81	5,701159726	0,03%
Skupni indeks	5162,5	5231,7	5229,4	5195,9	5194,1	5234,1	5153,7	5152,7	5255,9	5242,8	5205,28	38,79973654	0,24%

Tabela B.18: UnixBench meritve - navidezni stroj KVM

	1	2	3	4	5	6	7	8	9	10	Povp.	Std.dev.	Std.nap.
Dhrystone	12673,8	12695,9	12670,9	12676,5	12639,4	12578,3	12689,3	12657,6	12685,4	12684,9	12665,2	34,68682106	0,09%
Whetstone	1945,4	1945,1	1945,3	1945,2	1945,6	1945,6	1945,6	1945,5	1945,2	1945,5	1945,4	0,188561808	0,00%
Execl	5504,7	5502,8	5503,1	5515,1	5524,6	5518	5517,7	5522,5	5520,5	5520,3	5514,93	8,296860183	0,05%
Kopir.-1024B/2000B	3743,7	3722,1	3671,6	3794,7	3696,1	3741,7	3743,8	3747,9	3821,9	3840,7	3752,42	53,03796963	0,45%
Kopir.- 256B/500B	2432,9	2456	2398,9	2421	2375,3	2374,3	2348,6	2411,9	2442,5	2392,8	2405,42	33,79529224	0,44%
Kopir.-4096B/8000B	6650,2	6734,5	6846,3	6861,8	6930,1	6467	6758,9	6707,4	6646	6789,6	6739,18	132,3319211	0,62%
Uporaba cevi	8059,7	8049,4	8047,4	8060,1	8057,5	8059,7	8059	8063,7	8048,1	8049,4	8055,4	6,097540489	0,02%
Kom. med proc.	3026,4	2875,6	2621,2	2689,9	2785,2	2738,6	2882	2722,4	2887,7	2792,8	2802,18	118,0838384	1,33%
Kreir. procesa	4575,4	3884,5	4486,7	3357,6	4340,8	4119,5	4410,1	4305,4	4492,1	4464,5	4243,66	372,6894483	2,78%
Skripte - 1 sočasno	10013,1	10043,7	9990,4	10090,5	10091,3	10141,3	10162	10126,5	10137	10130,7	10092,65	58,61608141	0,18%
Skripte - 8 sočasno	8925,9	8863,7	8888,2	9023,1	8976,7	8989,5	8991,7	8983	9005,7	8967,1	8961,46	52,04054402	0,18%
Režija sist. klicev	7208,9	7259,6	7277,6	7039,4	7051,2	7004,6	7050,8	7030,3	7039,6	7019,6	7098,16	106,136853	0,47%
Skupni indeks	5382,4	5295,4	5308	5210	5320,3	5261,9	5337,2	5306,4	5363,7	5345,2	5313,05	50,17870841	0,30%

Tabela B.19: UnixBench meritve - navidezni stroj Xen HVM

	1	2	3	4	5	6	7	8	9	10	Povp.	Std.dev.	Std.nap.
Dhrystone	12686,9	12698,3	12698	12679,3	12699,8	12701,3	12691,2	12702,3	12685,1	12694,9	12693,71	7,7733805	0,02%
Whetstone	1945,7	1945,3	1945	1945,7	1945,7	1945,8	1945,8	1945,8	1945,5	1945,4	1945,57	0,266874919	0,00%
Execl	5168,3	5166,4	5158,4	5165,9	5175,1	5160,3	5165,8	5167,3	5165,6	5165,5	5165,86	4,477896331	0,03%
Kopir.-1024B/2000B	3751,8	3796,4	3802,8	3784,7	3801,2	3771,6	3691,6	3796	3810,9	3818,3	3782,53	37,34877806	0,31%
Kopir.- 256B/500B	2423,4	2445,3	2430,7	2460,1	2448	2477,2	2410,3	2328,4	2453,5	2425,9	2430,28	40,84512211	0,53%
Kopir.-4096B/8000B	6847,9	6708,1	6880,1	6849,6	6803,6	6882,1	6528,8	6587	6445,8	6700,9	6723,39	157,0122179	0,74%
Kom. cevi	7862,7	7856,7	7872,3	7845,3	7873,9	7865,7	7857,1	7844,5	7853,8	7871,4	7860,34	10,68313312	0,04%
Preklop med proc.	2348,3	2334,9	2340,1	2352,8	2340,1	2348,1	2329,3	2349,3	2347,3	2363,1	2345,33	9,592595524	0,13%
Kreir. procesa	4317,7	4260,3	4280,5	4360,4	4351,8	4322,7	4343	4292,2	4329,6	4320,9	4317,91	31,85287184	0,23%
Skripte - 1 sočasno	9809,8	9799,4	9800,5	9810,4	9806,1	9803,2	9791,4	9794,6	9790,6	9827,4	9803,34	10,9693715	0,04%
Skripte - 8 sočasno	8288,2	8270,3	8294,3	8293,6	8291	8258,4	8250,3	8289,3	8289,3	8286,4	8281,11	15,71287158	0,06%
Režija sist. klicev	7320,3	7324,6	7316,5	7328,3	7316,6	7317,6	7332,8	7308,9	7313,6	7263	7314,22	19,32941339	0,08%
Skupni indeks	5184,2	5174,7	5187,8	5199,1	5194,8	5196,6	5151	5151,2	5169,2	5183,7	5179,23	17,5145051	0,11%

Tabela B.20: UnixBench meritve - navidezni stroj Xen PVH

Dodatek C

Seznam slik in tabel

Slike

Tabele